AN ABSTRACT OF THE DISSERTATION OF

Atil Iscen for the degree of Doctor of Philosophy in Computer Science and Mechanical Engineering presented on May 14, 2014.

Title: Multiagent Learning for Locomotion and Coordination in Tensegrity Robotics

Abstract approved: ______________________________

Kagan Tumer Geoffrey Hollinger

Tensegrity structures are composed of pure compressional elements that are connected via a network of pure tensional elements. The concept of tensegrity promises numerous advantages to the field of robotics. Tensegrity robots are, however, notoriously difficult to control due to their oscillatory nature and nonlinear interaction between the components. Multiagent learning, a subtopic of artificial intelligence, provides the tools to address challenges of tensegrity robots. In multiagent learning, multiple entities simultaneously learn a task together while interacting with each other through the environment. This approach can be applied at two different levels: both to coordinate teams of multiple robots, and to control a single robot where different agents control different parts of the robot. In this work, we consider both cases, and apply two multiagent learning approaches (Reinforcement Learning and Evolutionary Algorithms) to tensegrity robotics problems at different levels. First, we take the model of an icosahedron robot, and use multiagent learning to control different parts. We use coevolutionary algorithms and fitness shaping to develop learning based robust rolling locomotion algorithm. After the locomotion aspect, we study multi-robot coordination using multiagent reinforcement learning and reward shaping methods. At this phase, we study reward shaping and develop methods to use reward shaping to improve the cooperation between multiple tensegrity robots. We explain how these results are simulated and validated by using physical tensegrity robots. Last, we explain how these results helped design and development of a tensegrity robot with rolling capability: SUPERBall.

Multiagent Learning for Locomotion and Coordination in Tensegrity Robotics

by

Atil Iscen

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented May 14, 2014
Commencement June 2014

Doctor of Philosophy dissertation of Atil Iscen presented on May 14, 2014.

APPROVED:

Co-Major Professor, representing Computer Science

Co-Major Professor, representing Mechanical Engineering

Director of the School of Electrical Engineering and Computer Science

Head of the School of Mechanical, Industrial and Manufacturing Engineering

Dean of the Graduate School

Atil Iscen, Author

ACKNOWLEDGEMENTS

I would like to thank my advisor Kagan for his guidance, patience and support. His support for me to pursue different directions gave me the opportunity to work on a new research field. The perfect balance between his lead and freedom to roam in different topics kept my motivation always up. In that sense, he became not only an advisor but more of a role model for my research career. Luckily, his encouragement for a dual major PhD led me to work with my second major advisor, Geoff. I would like to thank Geoff and my committee members for their help on fitting my dissertation between the two majors.

Outside the graduate school, I was lucky to have two more advisors at NASA Ames Research Center. During and after my internship at Ames, Adrian Agogino and Vytas SunSpiral played a big role in my research by introducing me to the topic of tensegrity robotics and the SUPERBall project. The encouragement I received from them, and their help whenever I needed resulted in a motivation for me.. With contributions of these two bright people, the collaboration between the research group at NASA Ames and OSU became the source of this dissertation.

The tensegrity group at NASA Ames played a big role in this dissertation. Without my colleagues, it would not be possible to get deeper in tensegrity robotics. I would like to thank Ken for his ideas and efforts with his robot ReCTeR, Jeremie for brainstorming simulation and controls ideas, Jonathan for his help in simulation of his design of SUPERBall. Thanks to Andrew, Brian and all other interns at IRG group at NASA Ames Research Center. Back at OSU, I would like to thank my colleagues from my research group for the welcoming environment and their help in all of the research papers that I have written during graduate school. Finishing this dissertation would not be possible without the help of Stephen, Mitch, Carrie, Will, Logan and Chris.

Outside the academic life, I would like to thank my friends Oguz, Tugrul, Murat and Ekin for making the life and graduate school an enjoyment. I would also like to thank countless number of roommates who assisted me during grad school and life in Corvallis and San Francisco. Finally, I would like to thank my parents for their patience and support for me while I moved to a different continent to follow my dreams. It would not be possible for me to reach my goals without their assistance.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

Multiagent learning is a widely studied subtopic of artificial intelligence. It is previously used in many real world domains such as robot coordination, air traffic, robot soccer, power grid, coordination of unmanned air vehicles and rovers. Multiagent learning consists of methods where multiple learning entities interact through the environment. The default fit for multiagent learning is naturally distributed problems such as multiple robot collaborating on a task. On the other hand, it is also widely used to perform distributed learning on a single entity problem. Multiagent setting allows us to divide the problem into smaller portions and to have different agents learn coordinate while learning to handle these smaller portions [88, 74]. Hexapod robots where different agents controlling different legs is an example to such an approach [6].

Learning in multiagent systems is a challenging problem, partly because each agent contributes to creating a dynamic environment in which all the agents operate. This issue becomes more pronounced as the number of agents increases. In addition to the dynamic environment, each agent also faces the credit assignment problem where it needs to determine how its actions contributes to the overall performance of the system. Both of these issues make agent coordination methods a key component of multiagent systems research [40, 24, 12].

The origins of many multiagent learning methods are single agent learning methods such as Reinforcement Learning (RL) and Evolutionary Algorithms (EA). For both RL and EA, the agents learn the good behavior using their interactions with the environment and a reward / fitness signal provided by the environment. When the problem is extended to cooperative multiagent setting, agents need to determine which set of actions are likely to lead to good behavior, without necessarily knowing what actions other agents will choose.

In cooperative multiagent problems, the team performance is not only sensitive to how well they perform the task, it depends on the agents' decisions with respect to other agents' actions. Since the team performance is what is desired to be optimized, the most trivial candidate for reward/fitness signal is the team performance. While being

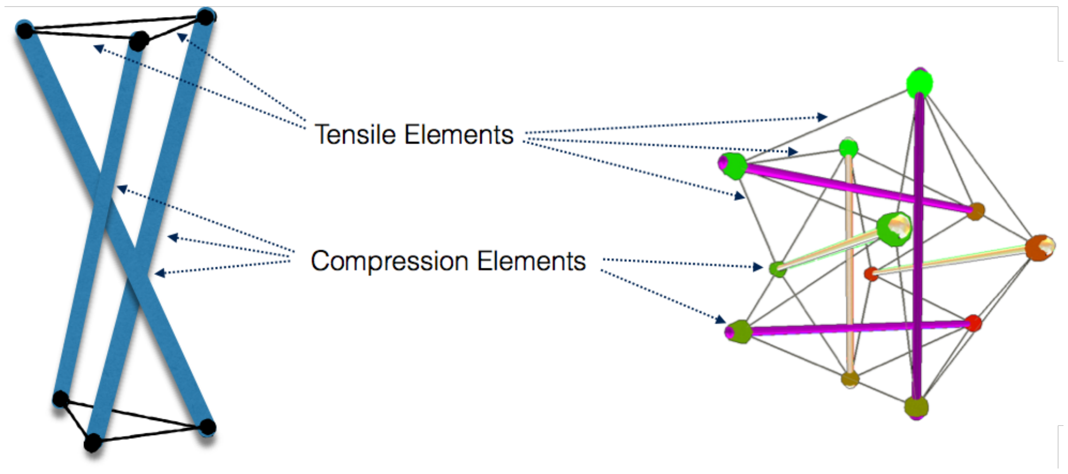


Figure 1.1: 2 different tensegrity structures with different complexity levels. The tensile elements connect the compression elements and keep the structure in internal balance.

the trivial option, the team performance is not necessarily a good reward structure. The individual agents cannot necessarily get feedback about their contribution to the success of failure of the team. A better approach is to craft the feedback for each agent according to the their contribution. This class of methods where the learning agents' feedback is individually crafted is called 'reward shaping' for Reinforcement Learning or 'fitness shaping' for Evolutionary Algorithms. Both methods aim to improve multiagent learning by providing rewards that are personalized depending on each agent's contribution to the team..

Through this dissertation, we will take the idea of multiagent learning combined with reward / fitness shaping and use it to address the problems of an emerging field of robotics: Tensegrity Robots. Tensegrity structures are based on a simple principle: the structure is composed of pure tension and pure compression elements. Axially-loaded compression elements are encompassed within a network of tensional elements; thus, each element experiences either pure compression or pure tension. Based on this simple principle, by increasing the number of members and by changing their stiffness, the structures can be arbitrarily complex and stiff. Figure 1.1 shows the simplest 3-bar tensegrity structure, and a more complex icosahedron tensegrity structure. The cables are in tension, the rods are in compression and the structure is in balance.

Since the tensegrity structure does not have any bending or shear forces that must be resisted, individual elements can be extremely lightweight. Moreover, the majority of the structure is composed of tension elements that are significantly lighter than the compression elements. A unique property of tensegrity structures is how they can internally distribute forces. As there are no lever arms, forces do not magnify against joints or other common points of failure. Instead, externally applied forces are distributed throughout the structure via multiple load paths, creating a system-level robustness and tolerance to forces applied from any direction. Thus, tensegrity structures can be easily reoriented and are ideally suited for operation in dynamic environments, where contact forces cannot always be predicted.

The concept of tensegrity structures offers a number of beneficial properties to robotics, including:

- **Lightweight**: Forces are aligned axially with components and shocks are distributed through the tensegrity, allowing tensegrities to be made of lightweight tubes/rods and cables/elastic lines.
- **Energy efficient**: Through the use of elastic tensile components and dynamic gaits, efficient movement is possible.
- **Robust to failures**: Tensegrities are naturally distributed systems and can gracefully degrade performance in the event of actuation or structural failure.
- **Capable of unique modes of locomotion**: Tensegrities can roll, crawl, gallop, swim, or flap wings depending on construction and need.
- **Impact tolerant and compliant**: Since forces are distributed upon impact, they can fall or bump into things at moderate speed. In addition, their compliance ensures that they do minimal damage to the objects they contact.
- **Naturally distributed control**: Characteristics of force propagation in tensegrities allows effective local controllers.

The last property is the most subtle but important. In “traditional” robots, distributed controls becomes messy due to the need to communicate global state information to all the controllers with high precision, and thus, often undermine the very promise of distribution. Fundamentally, this stems from the fact that a rigidly connected structure will

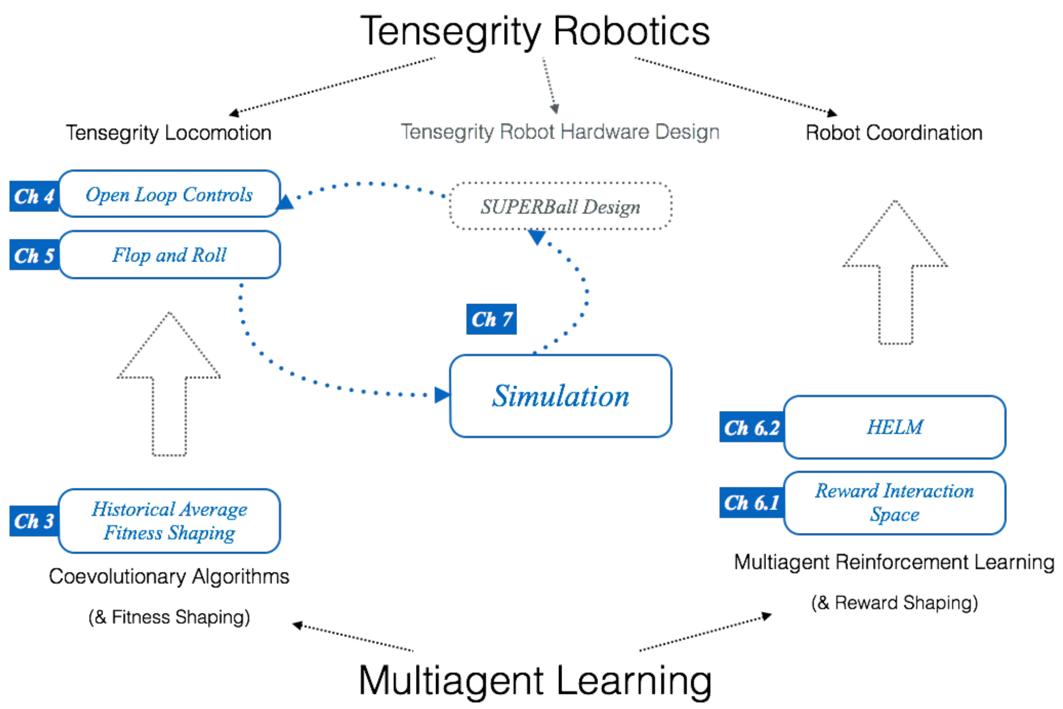


Figure 1.2: The chart flow to show where different chapters fit between tensegrity robotics and multiagent learning. In this thesis, we address two tensegrity robotics problems directly (tensegrity locomotion and multi-robot coordination) and one problem indirectly (hardware design). For locomotion we use coevolutionary algorithms and fitness shaping. For multi-robot coordination we use multiagent reinforcement learning and reward shaping.

magnify forces internally through leverage, and will accumulate force into joints. Thus, the actions of a local distributed controller can have disproportionate global consequences for the robot. These consequences can require a certain amount of global coordination and state management, undermining the value of the local controller. Tensegrity structures are different, due to the tension network, there is no leverage in the structure. Thus, forces diffuse through the structure, rather than accumulate in joints. As a result, actions by a local controller diffuse through the structure, integrating with all the other local controllers. While even one local controller will impact the structure globally, that impact is locality-relevant and not magnified via leverage. Thus, the structure enables true distributed control, because local actions stay (predominately) local.

We will use multiagent learning at two different levels to solve two different problems of Tensegrity Robotics: Tensegrity Locomotion (Chapters 4 and 5) and multi-robot coordination (Chapter 6). Figure 1.2 illustrates how we address the problems in tensegrity robotics using multiagent learning tools. For tensegrity locomotion, we will use multiagent learning to establish a unique mode of rolling locomotion. Since tensegrity robots are natural fit for distributed controls, we will use different agents to control different parts of a single robot. Tensegrity Locomotion brings many challenges to classical control methods. We will handle these problems using coevolutionary algorithms and fitness shaping. To improve state of the art in fitness shaping literature, in Chapter 3, we will present a new fitness shaping method that favors robust teams over optimal teams. Based on this idea, we will develop learning based controls for Tensegrity Locomotion.

For multi-robot coordination (Chapter 6), we will use multi-agent reinforcement learning and reward shaping in a higher level coordination problem in tensegrity robotics. As opposed to locomotion approach, different agents are in control of different robots. These agents will learn to cooperate to increase the performance of a team of robots working on a collaborative task. To improve the state of the art, we will first decrease information requirements for reward shaping methods. Second, we will define a decentralized task decomposition method based on reward shaping. We will study both methods in multi-robot coordination scenarios.

The contributions of this dissertation covers a broad area from coevolutionary algorithms and reinforcement learning to tensegrity robotics.

1. **Historical Average Fitness Shaping:** In Chapter 3, we develop historical average, a fitness shaping method for coevolutionary algorithms. The presented method favors robust solutions instead of optimal and speeds up learning using random sampling. At the end of learning, the resulting agents can handle noise or slightly different collaborators. This phenomena is shown with tensegrity locomotion.

2. **Open Loop Tensegrity Locomotion:** In Chapter 4, we show the feasibility of continuous rolling locomotion for icosahedron tensegrity robots. We coevolve controllers using historical average in the simulator and resulting open loop controllers provide rolling locomotion. Since these controllers are open loop, they don't require any communication or sensor information. These controllers are easy to apply to a hardware robot to provide required deformations that will result in

a rolling locomotion.

3. **Closed Loop Tensegrity Locomotion:** In Chapter 5, we present a tensegrity locomotion algorithm that uses contact sensors and desired direction as the only inputs. The algorithm exploits the symmetry of the tensegrity robot to learn faster. These controllers are coevolved using historical average. As a result, we show a directional and robust rolling locomotion algorithm for tensegrity robots. The robustness and feasibility of the algorithm is tested the simulator and a model of the SUPERBall.

4. **Interaction Space for Reward Shaping in RL:** In Chapter 6, we define interaction space to decrease information requirements of shaped rewards for multiagent RL, we present an algorithm to discover an interaction space given a problem. We present the results using multi-robot coordination problem.

5. **HELM:** In Chapter 6, we provide a decentralized task decomposition algorithm using shaped rewards and RL. These two concepts are tested in multi-tensegrity coordination domain to provide better learning performance in cooperation problems.

6. **Design loop for a tensegrity robot:** The design of a novel robot for an unknown locomotion is a chicken-and-egg style problem. In general, studies are conducted for locomotion of a known embodied robot or design of a robot for a known locomotion. Working on this cycle, we present the overall procedure to design a new prototype for the SUPERBall. Since both the robot and locomotion are novel to the literature, the design of the hardware and locomotion problems are closely tied together. In this dissertation, we show how simulation and evolutionary algorithms allows us to go back and forth between the locomotion problem and design problem. These results in locomotion assists designers and engineers to come up with hardware specifications that we present in Chapter 7.

The rest of the dissertation is organized as follows: Chapter 2 gives the necessary background and related work. Chapter 3 presents historical average fitness shaping algorithm. Chapter 4 and Chapter 5 presents two locomotion algorithms that uses historical averarge: open loop controls and flop and roll. Open loop controls is relatively easy to apply while flop and roll is more capable by using feedback from sensors. Chapter 6 presents two different tools for reward shaping in RL and apply these to multi-tensegrity coordination. Chapter 7 presents the simulator that we developed to study tensegrity robots, how it is validated using a previously developed underactuated robot: ReCTeR. This combination of software and hardware also leads to the next version of the prototype of SUPERBall that is currently under development. Last, Chapter 8 presents the conclusions that can be drawn from the dissertation.

Chapter 2: Background

The aim of this dissertation is to connect two separate research fields: multiagent learning and tensegrity robotics. Considering the difference between the two fields, we divided the background and related work into two main sections. We first start with defining tensegrity and the motivation behind tensegrity robotics. Tensegrity robotics is a relatively new research area, and since we are providing the first learning approach to the rolling locomotion, there is small amount of work that is comparable in terms of locomotion. The related work is mostly composed of other tensegrity robots in literature and the published papers about form finding techniques for tensegrity robots. In terms of multiagent learning and reward / fitness shaping, necessary background is given in Section 2.2. We describe two multiagent learning methods: multiagent reinforcement learning and coevolutionary algorithms. As the common point, credit assignment problem and reward or fitness shaping as a solution are described. In this context, related work is composed of other reward or fitness shaping methods.

2.1 Tensegrity concept

Tensegrity structures are a fairly modern concept, having been initially explored in the 1960's by Buckminster Fuller [22] and the artist Kenneth Snelson [73]. The word tensegrity is formed by combining the words 'tension' and 'integrity'. Tensegrity structures are based on a simple principle: the structure is a combination of pure tension and pure compression elements. Axially loaded compression elements are encompassed within a network of tensional elements, and thus each element experiences either pure compression or pure tension. Based on this simple principle, the structures can be arbitrarily complex and stiff by increasing the number of members and by changing their stiffness.

Based on this simple principle, a tensegrity structure can be as simple as 3 compression elements and 6 tensional elements. When the number of elements increase, the structure can contain dozens of compression elements connected by hundreds of tensional elements. The structure do not have to follow a uniform pattern, it is possible to construct structures with different geometric shapes. The sculptures based on tensegrities can be spheres, triangles or icosahedrons as well as structures with more complex shapes such as animal figures. Some tensegrity structures can also follow a repetitive pattern that allows the structure to be infinitely extended using smaller pieces of tensegrity structures.

Considering the base components of tensegrities, since compressional elements do not encounter any bending or shear forces that must be resisted, individual elements can be extremely lightweight. Compared to compressional elements, tensional elements are elastic cables that are mostly negligible in weight. The amount of tension that each component encounters is small compared to the overall tension, since the external forces are internally distributed to multiple components.

As there are no lever arms, forces do not magnify into joints or other common points of failure. Rather, externally applied forces are distributed through the structure via multiple load paths, creating a system level robustness and tolerance to forces applied from any direction. Thus tensegrity structures can be easily reoriented and are ideally suited for operation in dynamic environments where contact forces cannot always be predicted.

Based on the advantages of being lightweight and robust to external forces, the tensegrity concept was first considered in architecture. Parallel with its usage in architecture, the early research on tensegrity was first concentrated on the design and analysis of static structures [72, 7, 34]. The tensegrity principle was used in big structures such as tower or bridges as well as small structures such as toys or furniture [36].

The concept of tensegrity is also being discovered in biological systems, especially in individual cells.[28, 27, 29, 87]. The tensegrity principle is also observed in mammalian physiology [48, 43]. Emerging biomechanical theories are shifting focus from bone-centric models to fascia-centric models, where fascia is the connective tissues (muscles, ligaments, tendons, etc.) [64]. In the "bio-tensegrity" model, bones are under compression, and continuous network of fascia acts as the primary load path for the body. Inspired by this, the rest of the dissertation, the term ‘muscle’ is used to indicate tensional elements of the tensegrity robots. Based on the bio-tensegrity approach, literature contains recent research about simulating animal locomotion using the tensegrity model [50]. The strong connection between the concept of tensegrity and biological locomotion leaded to a new research field: ‘Tensegrity Robotics’.

2.1.1 Tensegrity Robotics

Traditionally robots are composed of bulky protective metal pieces that move using heavy motors and joints. They are mostly delicate and heavy. The tensegrity concept and the examples discussed above have a lot of differences from the classical robotics standpoint. As discussed in Section 1, tensegrity robotics offer many advantages such as being lightweight, energy efficient, robust to failures, compliant while performing unique modes of locomotion. Moreover, the structure being distributed, it is perfect match for decentralized control mechanisms.

Despite the desirable properties, tensegrity robots have remained mostly a novelty for many years due to properties that make them hard to control with traditional control algorithms such as:

1. **Nonlinear dynamics**: A force generated on one part of the tensegrity propagates in a nonlinear way through the entire tensegrity, causing shape changes, which further change force propagations.

2. **Complex oscillatory motions**: Tensegrity robots tend to have oscillatory motions influenced by their interactions with their environment.

Fortunately the combinatorial optimization capabilities of learning based controls is a natural match to these problems. Evolutionary algorithms can learn complex control policies that maximize a performance criterion without needing to handle the oscillatory motions and distributed interactions explicitly. Cooperative Coevolutionary Algorithms (CCEAs) enable different controllers that are distributed throughout the tensegrity to learn a cooperative task such as rolling locomotion. The set of the learned policies for these controllers form a unified policy that provides locomotion of the whole robot.

The idea of tensegrity robots became a research interest in recent years. The advantages and challenges discussed above presented an interesting research topic for the robotics community. For the first few decades, the majority of tensegrity related research was concerned with form-finding techniques [94, 46, 80, 66, 95, 58, 63, 41]. The problem of finding the correct configuration to deform a tensegrity robot to a specific shape is already a hard research problem due to the nature of the structure. To solve this problem, literature contains different approaches such as static, kinetic, density based methods [44, 47, 81].

In addition to deforming a given tensegrity structure, a second research area was about designing new tensegrity structures. Since tensegrity structures follow a pattern, coming up with a tensegrity structure for a specific task is another challenging problem. Evolutionary algorithms was one way to find new and non-regular forms [65, 66, 59, 93].

Despite the research on form finding, there are few hardware implementations for tensegrity robots. Koizumi et al. use a tethered icosahedron robot with pneumatic actuators to analyze base patterns for low energy rolling [39]. Rieffel et al. uses an icosahedron robot with vibrational motors analyzing forward motion with vibration [67]. Suitable for rolling locomotion, ReCTeR is the closest to the ideal icosahedron robot [15]. It is untethered, has six motors that control the lengths of six muscles. The robot is composed of a passive shell with 24 muscles. These 24 muscles are essential for the icosahedron. In ReCTeR, these 24 muscles are passive but the structure has 6 additional active muscles that change their length for deformation and locomotion.

SUPERBall is designed in a modular way so that each strut has 2 motors controlling two active muscles, and the struts can be used in different tensegrity robots other than icosahedron. Overall, the next prototype of SUPERBall will have an active outer shell with 12 active and 12 passive muscles, and it will be able to handle strong collisions and torque requirements for rolling locomotion in different types of terrains. Both of these robots, ReCTeR and SUPERBall, are presented in details at the chapter 7 of this dissertation.

2.1.2 Tensegrity Locomotion

The forms for tensegrity structures or robots were either biologically inspired or evolved. The ultimate goal behind the process of designing new tensegrity robots is the locomotion. Inspired by biological systems, tensegrity locomotion promises compliant and distributed robots that are close to human or animal locomotion. On the other hand, based on a simple principle, these tensegrity robots can be complex and hard to control. Since, deforming a stationary tensegrity structure is a research problem by itself, locomotion is a much more complex problem.

Tensegrity robots are capable of providing different forms of locomotion due to the diversity of the shapes of the structures [21]. When we look at simple irregular tensegrities with 3 or 4 struts, resulting locomotion can be similar to crawling [57, 56]. Biologically in-

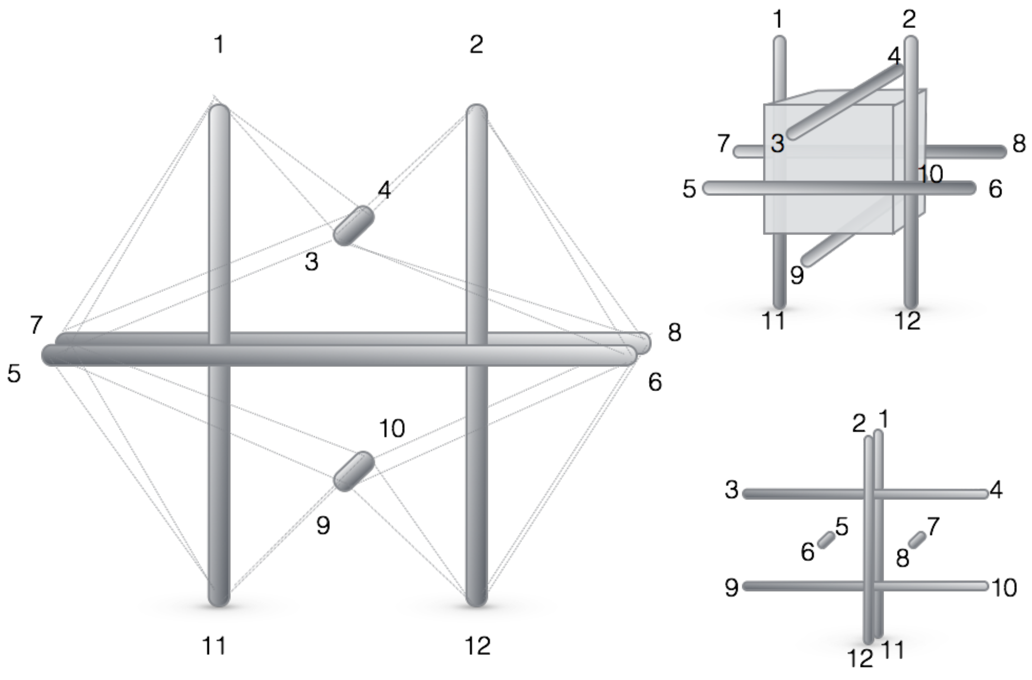
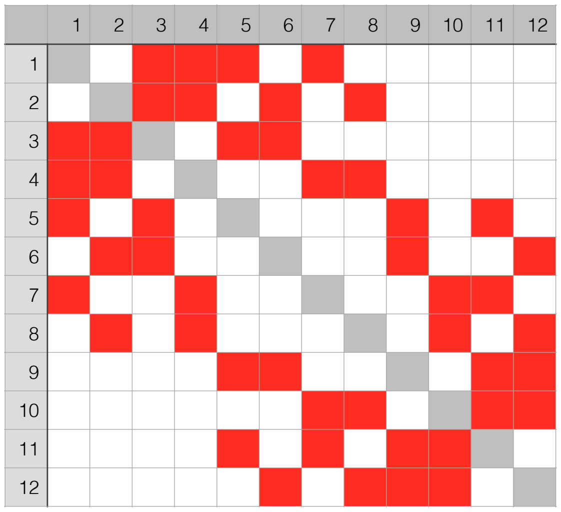


Figure 2.1: Icosahedron tensegrity with 6 rods and 24 muscles.

spired versions of crawling such as caterpillar's use more complex body and behavior[51]. One recent example is a tensegrity snake robot made of nested tetrahedral components which is capable of crawling over a wide range of terrains using a neurologically inspired distributed Central Pattern Generators (CPG's) control network [82].

Moving to tensegrity structures that are closer to sphere, icosahedron tensegrity robots provides additional advantages such as rolling locomotion and deployability. Figure 2.1 illustrates 6 rods of the icosahedron in their default orientation. These 6 rods have 12 endpoints, and 24 muscles connects these 12 endpoins. The connectivity matrix is given in Figure 2.2. In default orientation, all the muscles have the same length. The pattern of connection forms 8 equilateral triangles: $\widehat{1,3,5}$, $\widehat{1,4,7}$, $\widehat{2,3,6}$, $\widehat{2,4,8}$, $\widehat{5,9,11}$, $\widehat{7,10,11}$, $\widehat{6,9,12}$, $\widehat{8,10,12}$.

The structure is the simplest tensegrity to provide an overall shape close to a sphere. It can handle external forces and impact with the ground easily by deforming. As expected, the impact is diffused through the network of tensional elements. Moreover,



	1	2	3	4	5	6	7	8	9	10	11	12
1	▨		🟥	🟥	🟥		🟥					
2		▨	🟥	🟥		🟥		🟥				
3	🟥	🟥	▨		🟥	🟥						
4	🟥	🟥		▨			🟥	🟥				
5	🟥		🟥		▨				🟥		🟥	
6		🟥	🟥			▨			🟥			🟥
7	🟥			🟥			▨			🟥	🟥	
8		🟥		🟥				▨		🟥		🟥
9					🟥	🟥			▨		🟥	🟥
10							🟥	🟥		▨	🟥	🟥
11					🟥		🟥		🟥	🟥	▨	
12						🟥		🟥	🟥	🟥		▨

Figure 2.2: Connectivity matrix for 12 ends of 6 rods for the illustrated icosahedron robot.

the structure is easily collapsible to a star pattern by loosening tensional elements.

The literature contains few studies about active control of icosahedron tensegrities. Rieffel at al uses vibration frequencies for locomotion without rolling [67]. Controlled with different frequencies, these robots move on an even surface, but does not provide a rolling locomotion. Although it is possible to have crawling or vibrational locomotion with icosahedron tensegrity robots, rolling locomotion is the desired way to move an icosahedron. Due to its better usage of friction, it is more energy efficient. Moreover, the overall rolling behavior does not need a smooth surface. A rolling tensegrity robot can easily handle obstacles by taking advantage of its compliance. Despite these advantages, literature does not contain many studies on rolling locomotion. Hirai et al shows how to simulate an icosahedron robot for rolling locomotion [25]. Some studies analyze different surface patterns to roll an icosahedron tensegrity with pneumatic actuators [71, 70, 39]. Not only icosahedron tensegrities, but also the whole field of active control of tensegrities is still fairly new. A recent review [85] shows that there are still many open problems in actively controlling tensegrities.

Compared to the related work in this field, to the best of our knowledge we provide the first continuous rolling locomotion algorithm for a tensegrity robot. We provide both the first open loop control and directional rolling algorithm. On the other hand, literature of rolling robots does contain more research outside tensegrity robots. The closest to an icosahedron tensegrity is RATS [86]. The robot is spherical and contains 12 legs. The similarity between RATS and a icosahedron tensegrity is based on number of legs and their placement. The positioning of the legs are exactly same as the end of the rods for icosahedron. Moreover, the footprint of the two robots are similar. Although resulting locomotion is similar, the difference comes from the way locomotion is established. RATS has actuators that apply force perpendicular to its surface, but an icosahedron tensegrity robot moves by deforming itself using its shell muscles. Considering nonlinear controls, designing a locomotion for a tensegrity robot is a much harder problem.

Both icosahedron tensegrity robots and RATS roll on these 12 end points that can be considered as foot. The advantage of such a locomotion is the ability to interact with the ground at discrete locations. These legs take advantage while walking on an uneven or discontinuous terrain (such as small obstacles). Contrary to wheeled locomotion or spherical robots, stepping or hopping on these obstacles is possible for RATS or icosahedron tensegrities. In this sense, the locomotion presented in this dissertation is similar to legged locomotions such as humans or animals. On the other hand, rolling provides additional advantages to legged locomotion such as the robustness against balancing problem that legged robots suffer. Moreover, rolling is symmetric for any orientation and it is efficient compared to lifting and moving legs for each step.

For spherical robots, the literature contains multiple ways to provide rolling locomotion. Moving center of mass is one way to move the robot [9]. Despite spherical robots, tensegrities are much harder to use this strategy. First, the robot lies on a triangle because of icosahedron shape, second moving the center of mass of a tensegrity is a complex control problem by itself. In our approach the robot rolls by changing the center of mass to roll, but this is the consequence of the learning algorithm, not the goal. Another strategy to roll for spherical robots is to use angular momentum of internal structure [8]. This strategy is not feasible for tensegrity robots since we are trying to take advantage of the lightweight and compliant nature of tensegrities. Instead we will keep the robot lightweight and make the robot learn establish rolling locomotion by deforming itself.

2.2 Multiagent Learning

Multiagent Learning is the core element through this dissertation and it is used in form of different algorithms to address different problems in Tensegrity Robotics. While, chapters 4 and 5 use coevolutionary algorithms to control different parts of a tensegrity robot to achieve locomotion, chapter 6 uses Reinforcement Learning to achieve multi-tensegrity coordination. To provide necessary background, this section presents Reinforcement Learning, Evolutionary Algorithms, the concept of reward shaping (and fitness shaping for Evolutionary Algorithms) for multiagent learning.

An agent is an autonomous entity interacting with the environment through its actuators according to the information sensed by its sensors. A Multiagent System (MAS) can be defined as an environment with more than one agent interacting with each other through the environment. In a MAS, the agents should have some independence from other agents in the sense that they may not know everything about the environment including other agents' internal states. To distinguish MAS from a single agent system, another constraint is on the amount of communication that the agents have. Stone and Veloso states that a system with full communication can be considered as single agent by providing full state of the environment to a centralized decision mechanism [74].

Some domains can be classified as 'multiagent' by definition, such as multiple rovers on a discovery mission. When it is not possible to control all the agents through a centralized mechanism, each rover is controlled by an independent agent. On the other hand, some problems can be transformed to a multiagent problem by having different agents controlling different parts of the system. For example, multiple agents controlling different legs of a six legged robot is also a multiagent problem.

Multiagent problems can be cooperative, competitive or mixture of the two. In this dissertation we concentrate on cooperative problems where a team of agents have a common goal. Multiple robots that explores an area in an environment is a classical example to this type of problems.

In a single agent system, the agent is the only entity that acts and change the state of the environment. On the other hand, in a multiagent system, the state of the environment depends on the actions of all the agents interacting with the environment. The environment becomes more and more dynamic with increasing number of agents. It can give many different responses to the same action according to other agents' behaviors.

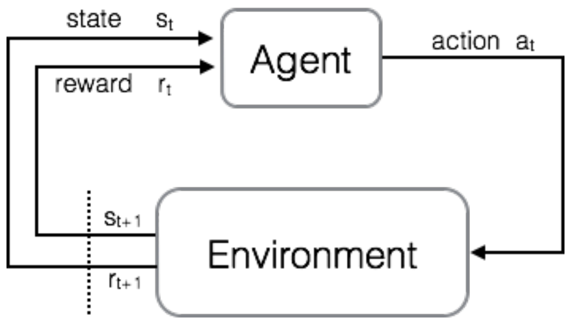


Figure 2.3: Reinforcement Learning: The interaction between the agent and the environment.

Considering the scenario of concurrent learning, where more than one agent tries to learn at the same time, the changing behavior of the agents adds one more level of complexity to the learning problem. Although there are many algorithms that are proven to converge to optimal behaviors for single agent problems, it is shown that using these algorithms for a MAL problem does not always lead to convergence to optimum even if the agents can perceive all the other agents' actions [16].

2.2.1 Reinforcement Learning

Reinforcement Learning (RL) is a specific type of learning method, where an agent learns from its interactions with the environment [76]. In RL, the agent is expected to learn the optimal behavior using the rewards given by the environment as feedback. At a given timestep, the agent gets the state information from the environment, then decides on an action. According to the state and the action of the agent, the environment returns the new state and reward. [35]. Reinforcement Learning has been extensively used in multiagent learning, even if many of its theoretical guarantees often do not hold [74].

The task is defined in terms of Markov Decision Processes (MDPs) specificied by the tuple (S, A, T, R). The agent observes a state $s \in S$ that is a vector of state variables. It decides on an action $a \in A$. Reward function $R : S \times A \rightarrow \mathbb{R}$ maps state and action to reward value. Transition function $T : S \times A \rightarrow S$ maps the state and action to a new state. Given these definitions, the policy of the agent $\pi : S \rightarrow A$ defines which action to take on a given state.

The Q value of a state action pair, denoted by $Q_\pi(s, a)$, is the estimation of the sum of all the rewards that one agent would get by taking action a at state s and following policy π. Through the learning process, the agent improves the estimation of Q values and the policy to increase performance for the given problem. In action value learning algorithms such as q-learning and sarsa, the agents learn Q values assigned to each state action pairs. In the sarsa algorithm, this update is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r_t + \gamma Q(s_{t+1}, at + 1) - Q(s_t, a_t)\}$$

where α is learning rate and γ is discount factor. Using these updates, the Q values converge to discounted values of the best action for the successor state, and the policy becomes selecting the action that gives the best Q value for every state.

In the definitions given previously, the state s is a discrete entity, but in many real world problems, the state variables are continuous which requires discretization. Even when the states are discrete, the number of states can be impossible to handle. Moreover, generalization between similar states can speed up learning significantly. Because of these reasons, RL algorithms are usually coupled with different types of function approximations [13].

2.2.2 Reward Shaping in Multiagent Reinforcement Learning

In a multiagent setting, reinforcement learning setup that we described changes significantly. Since multiple agents interact with the environment at the same time, the reward provided by the environment depends on other agents. This setup is shown in figure 2.4. Each agent learns independently using the state and the reward received from the environment. As opposed single agent RL, the environment becomes highly dynamic.

The direct application of RL to multiagent systems contain different approaches [26, 69]. These methods mostly contain different extensions or modifications to RL according to the needs of a multiagent scenarios, and they have both advantages and disadvantages. The benefits of MARL algorithms are related to speeding up learning using agent interactions like communication, coaching or imitations [62, 77]. On the other hand, disadvantages are the curse of dimensionality and highly dynamic environment as discussed before. The different types of MARL methods and their drawbacks can be

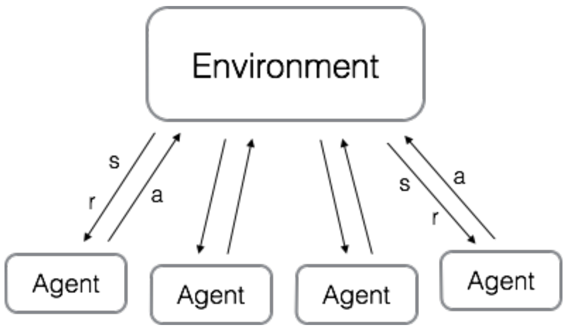


Figure 2.4: Multiagent Reinforcement Learning: The interaction between the agents and the environment.

found in these details surveys by Panait et al [53] and Busoniu et al [12].

One problem of direct application of single agent methods to multiagent learning is the reward provided by the environment. In a RL setting, learning is based on the reward that the agent gets, but the reward that the environment provides depends on all the teammates. This problem is called structural credit assignment and it is addressed by reward shaping methods [45, 49, 19, 42].

Credit assignment problem is independent of the learning method that is used. Reinforcement Learning or Evolutionary Algorithms, both encounter the same problem when there is a team of agents that learn a cooperative task. During the learning process, the agents sense the environment, take an action and receive a reward from the environment to improve their policy. Looking from each agents perspective, the reward received is highly dependent on other agents. A particular agent can take the best possible actions in all scenarios and still get a low reward because its teammates. Using this reward, a good or even the best policy can be treated as a bad policy.

As an example, lets consider a team of agents that learn how to play soccer. If the reward received by the agents is a team reward, such as the outcome of the game, a good player can easily be associated with a bad reward. If we take the example out of the multiagent learning context, if we could take the best soccer player and make him/her play with a really bad team, the outcome of the came can easily be against the team that the best player is playing with. The judgement according to the outcome of the game and the reward associated with it does not truly show the performance or the potential of the players of the team.

To solve such a problem, the reward provided to each agent should be crafted according to their performance and their contribution to the team affecting the outcome of the game. Ideally, in the soccer scenario, the best player would receive a high reward although the the outcome is a heavy loss due to team performance.

One solution to the credit assignment problem is Difference Rewards [2]. Difference rewards calculates the contribution of each team member by comparing the reward of the team with and without the agent. It is used with both Reinforcement Learning as well as Evolutionary Algorithms under the name 'Difference Evaluations'.

Before giving the mathematical definition, let us discuss two natural (unshaped) rewards that can be provided to an agent operating in a team. First, one can provide the "Global" reward (G), which represents full system performance, and second, one can provide a "Local" reward (L) which represents an agent's direct contribution to the global reward. One can see that in large systems providing the global reward becomes problematic as the impact of an agent's actions are obscured by the actions of all the other agents [2]. Similarly, one can deduce that having each agent simply aim to pursue its own contribution does not lead to coordinated behavior.

To overcome the problems of these two rewards, the Difference Reward (D) has been developed to provide a shaped reward that is more learnable than global reward and more factored than local reward [4]. It is defined as:

$$D_i \equiv G(z) - G(z - z_i + c_i) \ , \tag{2.1}$$

Where first term $G(z)$ is the global reward of the state z, second term $G(z - z_i + c_i)$ is the global reward of the system where the agent i is taken out and is replaced by a default action c_i. Subtraction of the second term from the first term gives the effect of the agent on the system.

Difference rewards are shown to provide better coordination and faster learning in different domains[84, 1, 83, 37]. On the other hand, it is not directly applicable to every domain because of different requirements of the definition of the difference rewards. In chapter 6, we address one of these problems: information needs of the difference rewards formulation. We then use proposed improvements on apply RL and difference rewards for multiagent learning in multi tensegrity coordination.

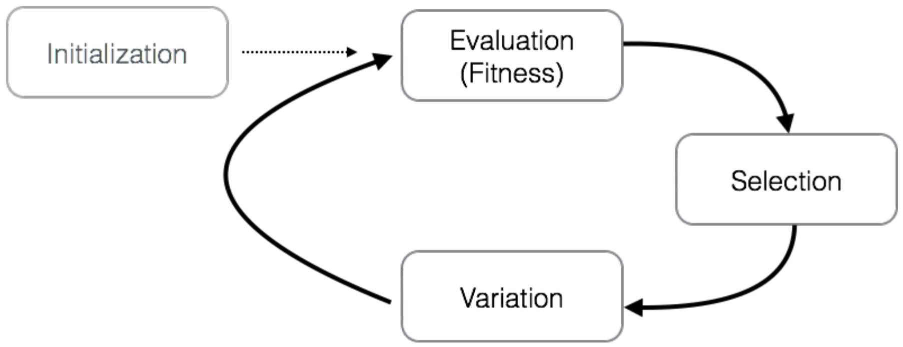


Figure 2.5: Evolutionary algorithms: the loop of evaluation, selection and variation

2.2.3 Evolutionary Algorithms

Evolutionary Algorithms (EA) is a class of optimization algorithms that is inspired by biological evolution [20, 18, 23]. Evolutionary algorithms has shown to success in problems from diverse fields such as robotics, physics or economics. An evolutionary algorithm is composed of different phases such as initialization, evaluation, selection, mutation, reproduction. Figure 2.5 shows the main cycle for a typical evolutionary algorithm.

For a given problem, an Evolutionary Algorithm starts with a population of candidate solutions, these candidates are evaluated according to their fitness. According to their fitness, the best candidates are selected for reproduction, and the worst candidates are eliminated. The selected candidates reproduce using different strategies and new candidates replace the eliminated ones. The cycle continues with evaluation of the new candidates.

Each iteration of the cycle is considered as one generation. During later stages of the cycle, the algorithm produces better candidates since they are based on the best solutions that are encountered so far. The literature contains many different evolutionary strategies for the single agent problems such as different selection, mutation or reproduction operators.

In a multiagent setting, natural extension of evolutionary algorithms is called coevolutionary algorithms [61, 60]. In coevolutionary setting, the agents interact with each other, but each agent has a separate gene pool and evolve independently except the evaluation part. Since the performance of each agent is affected by other agents, the

Algorithm 1: An Example Evolutionary Algorithm

```
Data: Population of $n$ candidates
for $m$ generations do
    forall the candidates do
        evaluate(candidate);
    end
    order the population;
    eliminate last $k$;
    reproduce $k$ elements;
    variation using mutation or crossover;
end
```

evaluation is done using the other agents that are evolving concurrently.

Depending on the multiagent problem, coevolutionary algorithms can be cooperative or competitive [52]. Since our work on Tensegrity Robotics (both locomotion and coordination) is about agents that are learning to cooperate as a team, we are using Cooperative Coevolutionary Algorithms (CCEAs) [91]. The default extension of EAs to CCEAs is illustrated in Figure 2.6. Each agent is evolved separately in its own population, but the evaluation part requires forming a team of agents from each one of the populations. The fitness is assigned to the agents according to the performance of the team that they were part of.

Considering the soccer example, we coevolve 11 agents as a team to play soccer. For each player, we have a separate pool of candidates, they reproduce and mutate indepent of other 10 agents. The additional work with CCEAs arise during the evaluation phase. Since there is not any regular way to evaluate a single soccer player, we have to form a team of 11 players to see the outcome. To solve this problem, a team is formed by selecting one of the candidates from each pool. All these 11 candidates are evaluated according to the performance of the team. Since the selected teammates affect the fitness of the candidate, next section discusses the problem with evaluation in CCEAs and the strategies that the literature contains to address the problem.

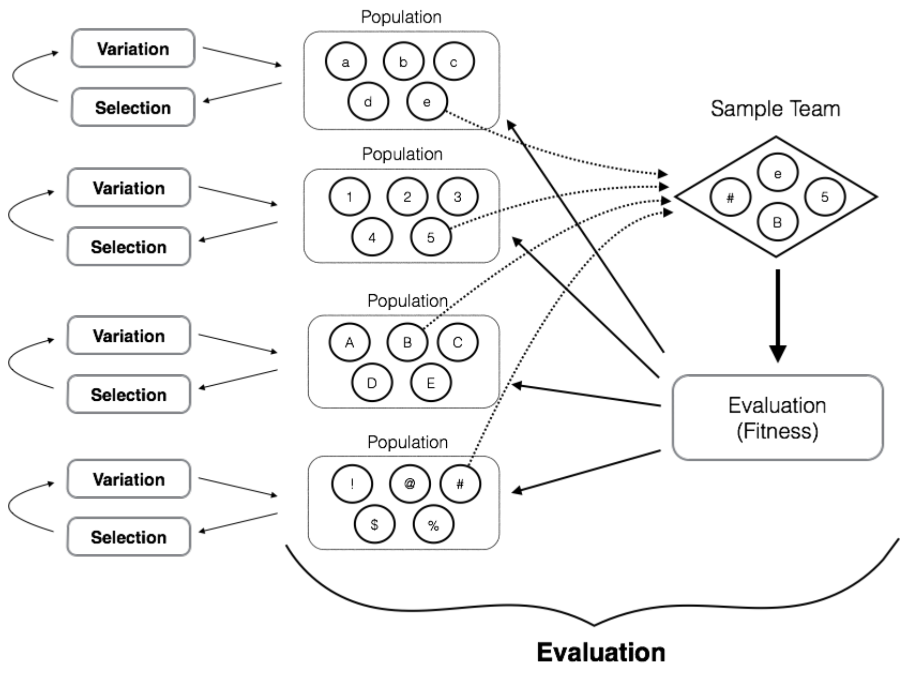


Figure 2.6: Coevolutionary algorithms: Each agent evolves in separate pools, but evaluation is done by forming teams.

2.2.4 Fitness Shaping in CCEAs

From cooperative coevolutionary algorithms perspective, the same problem of credit assignment that we see in multiagent RL arises in a different form [90]. CCEAs contain two levels of credit assignment problem. The higher level credit assignment problem is selecting the teammates for the candidate. As described in multiagent reinforcement learning, the outcome of the experiment highly depends on the teammates of the agent. The lower level credit assignment problem is the same as the multiagent reinforcement learning setting: once the candidate is tested within a team and the outcome is known, how are each individual credited according to the team performance.

The higher level credit assignment is the selecting best teammates for the candidate during evaluation phase. Since the best (or the most compatible) teammates are not known, there are different strategies to extract highest potential for each candidate. One of these strategies is 'Hall of Fame' [68]. The idea behind Hall of Fame is to test each

Algorithm 2: An Example Cooperative Coevolutionary Algorithm

```
Data: Population of $n$ elements for each agent
forall the $candidates \in Populations$ do
    form a team including the candidate ;
    evaluate(team);
    assign credit to the candidate ;
end
forall the $Populations$ do
    order the population;
    eliminate last $k$;
    copy first $k$ to last $k$;
    set score of last $k$ to $MIN$;
    mutate last $k$;
    clear history for last $k$;
end
```

candidate with the best team to see their true potential. On the other hand, the best team is not known by the time of learning. Instead, the agents are tested with the other candidates that have shown the best performance so far. Colby and Tumer combined this idea with the difference rewards to use each individual's contribution while being tested with the best team known [17]. One problem with Hall of Fame can arise during early learning. Since the hall of fame team is the approximation of the best team, during the early stages of learning it can favor some agents that are more compatible with a specific type of teammates. If the approximation is not good, the learning will be biased towards a specific local maxima.

Another approach is using lenient agents. Panait et al proposed to test each candidate with all possible teammates at the current generation, and to take the maximum score for each candidate as its fitness value [54, 55]. Instead of being tested with one possible team, they are tested with multiple teams that are formed using the candidates for other teammate positions. In this setting, the search is less biased, and it is easier to overcome local maxima. On the other hand, number of experiments per generation increases exponentially according to the number of agents and the size of the populations.

To address this higher level fitness shaping (or teammate assignment) problem, we design Historical Average fitness assignment algorithm in the next Chapter. We discuss both the differences from these two methods and different aspects that it brings to learning locomotion for tensegrity robots.

Chapter 3: Historical Average Fitness Shaping

The first step to our thesis statement, and SUPERBall, is to establish the rolling locomotion for icosahedron tensegrity robots. In chapters 4 and 5, we will present two approaches for tensegrity locomotion that we developed using coevolutionary algorithms and fitness shaping. As we stated in the background (Chapter 2), in CCEAs the agents evolve in separate pools while being evaluated by teams formed between agent evolving in these different pools. Each agent is evaluated with multiple teams and its survival depends in its performances within these teams. As a result, learning is biased towards agents that perform good with different collaborators. This bias results in finding robust solutions over optimal ones [92]. For many problems in robotics, these robust solutions are more important than the best solutions that are sensitive to small changes in the environment. In that sense, CCEAs are a clear fit for robotics.

On the other hand, for each generation, the computational cost of evaluating each agent depends on number of agents and size of the populations. As a result, the number of evaluations becomes exponentially higher in more complex problems. To reduce this cost while keeping benefits of CCEAs, we define Historical Average fitness shaping method. Historical Average fitness shaping still results in robust solutions by favoring the agents that can perform good with different roommates. The number of evaluations is kept lower using random sampling teams. Over the course of coevolution, the best candidates that survive multiple generations are evaluated with more teammates to increase the accuracy of random sampling.

In this chapter, we will first explain the benefits of CCEAs and robust solutions, discuss random sampling method to form the teams, define historical average fitness shaping and present empirical results with tensegrity rolling locomotion experiments. The rest of the chapter is organized as follows: Section 3.1 explains the robustness provided by the coevolutionary algorithms. Section 3.2 analyzes the effect of random sampling on CCEAs. Section 3.3 presents 'Historical Average' and Section 3.4 illustrates empirical analysis of the method.

3.1 Average Fitness Assignment and Robustness

Chapter 2 gave a brief overview of coevolutionary algorithms and the purpose of fitness shaping. As discussed before, CCEAs allow to learn a team goal for multiagent systems, but there are two levels of credit assignment problem. Higher level problem is forming the right teams to test and lower level problem is to distribute the credit to team members after testing the selected team.

Current methods to address these problems (leniency, hall of fame, difference fitness) have their own trade-offs between benefits and disadvantages. For example, the standard way to design tensegrity locomotion as a learning problem is to have multiple agents to control different parts of the robot and the reward is given according to the end performance after a fixed time window. This gives us a time extended multiagent problem with a delayed reward. In a multiagent problem with a delayed reward, calculating the difference fitness is not straightforward. Considering one test run, the performance of the team without a specific agent cannot be directly calculated. Using approximations of the difference fitness functions is an ongoing research topic with success [17].

At the higher level, the problem is related to the teammates that each candidate is tested with. In that sense, there are multiple options such as number of teams to try, selection of the teammates and the fitness assignment according to multiple tests. One of these options is using lenient agents that is also named as optimistic credit assignment [89, 90]. The idea behind lenient agents is to forgive poor performances caused by bad teammate combinations. Each candidate is tested in multiple teams and the best score of each agent is taken as its shaped fitness. All the candidates in populations for different agents (since agents are coevolving, they have separate gene pools) receive the maximum score that they can reach using current potential teammates.

Similar to leniency, hall of fame searches for the best performance of each candidate. To evaluate the candidate, the algorithm selects the best team of the previous generation and looks how that particular agent can perform in this team. This strategy used promotes a candidate according to a single performance as opposed to compatibility with different candidate teammates.

Without leniency or hall of fame, Wiegand and Potter [92] states that cooperative coevolutionary algorithms might prefer robust solutions to high quality solutions. Panait et al [55] addresses this problem with leniency by promoting maximum potential performance over robustness. This notion is previously explained using an optimization problem using 2 variables. The search space contains one large local maximum that is easy to converge and a global peak that is harder to reach. It was shown that, CCEAs tend to converge to the suboptimal area while lenient learners tend to converge to the optimal point.

Optimizing a robotics locomotion algorithm, we have different motivations than finding the best solution. The robots experience both sensing and actuation noise that might cause problems to the algorithms that prefers the best possible performance over robustness. Moreover, the motivation behind the study of tensegrity locomotion is not particularly finding the best possible locomotion. Instead, we work on showing that rolling locomotion is possible for icosahedron tensegrity robots, which makes applicability and robustness have a higher priority over the best result.

For problems where robustness is important, the fitness shaping method should promote the compatibility with different teammates as well as the best performance. We might prefer a solution that would give similar performance with slight changes of initial conditions, instead of a great solution that might not work with small amount of noise. One strategy to promote compatible solutions is to take average score (instead of maximum) of a candidate while testing it with random teams (instead of the best team). This is actually the default setup for CCEAs.

3.2 Random Sampling

The problem with the average fitness rises with increasing number of agents and increasing population size. If the agent is tested with all possible teammates within that generation, the number of simulations per generation increases up to size of the population to the power of number of agents. For a reasonable multiagent problem with 10 agents, a CCEA with the population size of 10, the number of experiments per generation easily increases to 10^{10}.

To address this problem random sampling is a commonly used method. Random sampling decreases the number of evaluations between two generations. Especially, while using average performance as the fitness, it is possible to get an approximate with slightly less number of evaluations. A good sampling size depends on the problem and the generation. Small number of evaluations can result in a higher error margin. Higher sampling values will result in a closer approximation of the average fitness with all possible teammates, but it increases number of evaluations per generation, therefore computational complexity of the algorithm.

A good sampling size depends on the problem and the generation. Small number of evaluations can result in a higher error margin. Higher sampling values will result in a closer approximation of the average fitness with all possible teammates, but it increases number of evaluations per generation, therefore computational complexity of the algorithm.

3.3 Historical Average

The number of samples per generation is a critical number for random sampling. It results in a tradeoff between good approximation and computational cost. To take advantage of this tradeoff we developed Historical Average, a fitness shaping method that provides better approximations for better candidates without increasing computational cost. In evolutionary algorithms, fittest candidates survive for the next generation. During future generations, each team member's gene pool contains different candidates. The old candidates that survived previous generations are now evaluated with new team members. If these surviving agents are not modified with a mutation, they are actually evaluated with more teammates over the course of multiple generations. Historical Average proposes to use the history of evaluations that these surviving agents collect.

Algorithm 3 shows the general flow of one generation of CCEA with Historical Average. Lines 1-10 show the random sampling for selecting teams. For each teammate of the selected teams, the team score is added to each team member's history (line 8). While assigning the fitness, each agent's fitness is selected as the average of all the scores in the history (line 13). Lines 15-19 are the elimination and breeding phase of coevolution.

For example, consider a system with 3 teammates and pool size of 10 (30 candidates). Instead of all possible team combinations (10^3), we choose to use random sampling with n evaluations. After n evaluations, let's say that the fittest 5 of each pool survives for the next generation (15 total). The bottom 5 of each pool are replaced with new candidates and these 30 candidates are evaluated with another n evaluations using randomly selected teams.

For each random team, the chance of a specific candidate being selected is $\frac{1}{5}$. After

Algorithm 3: Historical Average fitness shaping for CCEAs

```
Data: Population of $n$ elements for each agent
for $i$=1..15 do
    randomAgent team $\leftarrow \emptyset$ ;
    forall the Populations do
        randomTeam $\leftarrow$ randomAgent;
    end
    score = evaluate(randomTeam) ;
    forall the agents $\in$ randomTeam do
        agent.history $\leftarrow$ agent.score ;
    end
end
forall the Populations do
    forall the agents do
        agent.fitness = average(agent.history) ;
    end
    order the population;
    eliminate last $k$;
    copy first $k$ to last $k$;
    set score of last $k$ to $MIN$;
    mutate last $k$;
    clear history for last $k$;
end
```

n random sampling, each candidate will be evaluated approximately $\frac{n}{5}$ times in average. With historical average, previous top 5 that survived for this generation will use their previous evaluations together with the new ones to take the average of approximately $\frac{2*n}{5}$. For a candidate that survives k generations these size of its 'history' becomes $\frac{k*n}{5}$. The fitness of that candidate is the average of those $\frac{k*n}{5}$ evaluations during last k generations. Algorithm 3 illustrates this process step by step.

The purpose of taking average of these evaluations is to approximate average compatibility of that candidate with other possible teammates. For these best candidates that survives multiple generations our approximation gets better with an increasing size of history. Although we use a small number of evaluations per generation, longer these best agents survive, better their fitness approximation gets.

One concern with historical average can be the fact that a good and long surviving candidate is still judged with the teammates that coevolutionary algorithms had during the early generations. One can argue that CCEA produces better teammates that will provide better results. On the other hand, since the size of the history increases by $\frac{k*n}{5}$ at each generation, the effect of the evaluations using past (and possibly worse) teammates decreases over time. In addition, at every generation, these surviving candidates will have offsprings that can possibly perform better with the new generation teammates. The higher evaluation scores that these offsprings receive will eliminate their ancestors that survived many generations.

As an overview, while using Historical Average in CCEAs:

- Average score allows more robust solutions.
- The number of evaluations per generation decreases using random sampling of possible teams.
- The surviving candidates keep their history of evaluations. They grow a larger history to take average of, therefore they have a better approximation of their fitness.
- Each candidate receives a shaped fitness according to both current possible teammates and their past.

From biological inspiration perspective, if we analyze the resulting behavior, surviving candidates use their past experience to better approximate their fitness. On the other hand, past experience relies on past teammates that are possibly worse than current teammates. If there is a big difference between current and old teammates, new offsprings of these experienced candidates obtain better average since they are only experienced with the new teammates. As a result, new candidates eliminate old experienced candidates and the cycle continues as expected. As a summary, the whole process reminds the interactions between different generations of human beings. The good candidates survive longer as usual. Surviving candidates use their experience to shape their fitness. Since their teammates evolve and change, if their teammates get better, they cannot compete with new offsprings that are more compatible with new generation teammates.

There are open ended questions about historical average such as the effects of the amount of random sampling, the average history size before and after convergence, the variation within gene pools and the robustness of the produced results. Through the next chapters, we use historical average to solve the tensegrity locomotion problem and we investigate the performance and these open ended questions.

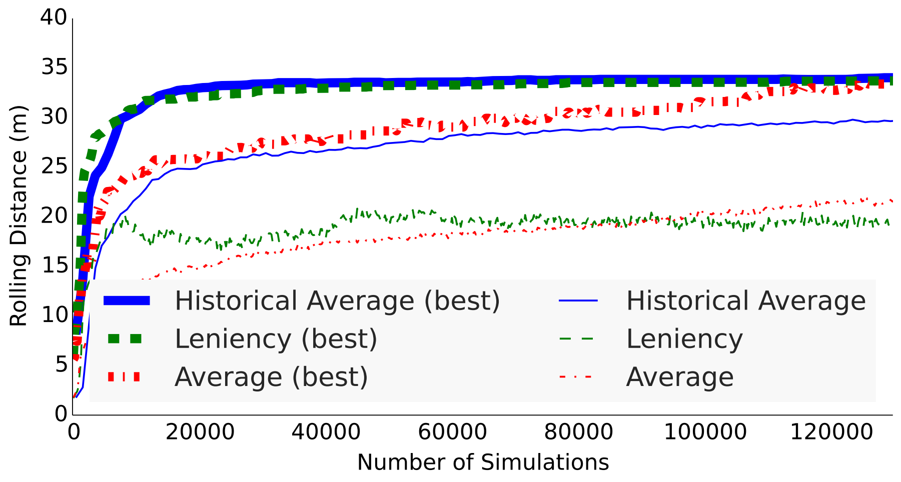


Figure 3.1: The comparison of 3 methods of fitness assignment when used with random sampling. Both the score of the best team and the average score per generation are shown. Although leniency reaches to the best team faster, the average score per generation is lower. The best score of historical average reaches the same score, and the average score per generation is much higher than leniency. Solutions found by historical average not only reaches the same best score, they are also more compatible with variations of its teammates.

3.4 Empirical Analysis

First, we compare historical average, lenient learners and a regular coevolutionary setup. We use tensegrity locomotion problem that we are studying in Chapters 4 and 5, but any background about the problem is not needed to analyze the result that we present here. As we discussed we are expecting to see that historical average produces more robust solutions that can work with slightly modified teammates. One way to empirically validate this conclusion is to look at the average score per generation. At every generation, agents are tested with random teammates multiple times. Each population contains

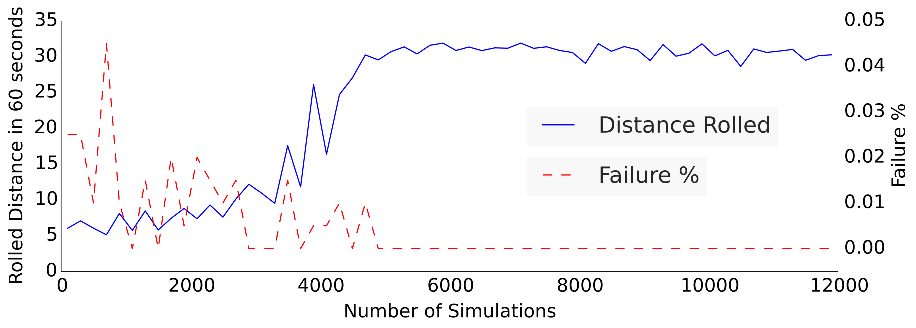


Figure 3.2: Learning curve and failures over time during the learning session for signals of a complexity of five and a period of four seconds. As a side result, the percentage of the policies that were failed to stay in reasonable limits are shown in the second line. While learning optimizes distance rolled, historical average picks safer policies that results in zero failures after convergence.

slightly modified versions of the best candidates. As a result, the average score of that generation shows the candidates' ability to work with slightly modified teammates. On the other hand, the maximum score that we obtain at each generation is the score of the best team that we obtained so far.

Figure 3.1 shows both the best and the average score per generation for 3 algorithms. We test standard CCEA, lenient learners and historical average. All 3 algorithms use random sampling of 50 teams per generation. When we look at the maximum scores reached by 3 algoritghms, they all converge to the same policy. Leniency converges faster since it favors best solution over robustness. On the other hand, if we look at the average scores per generation, leniency converges to a lower point than average fitnesses. This means that the the solutions that historical average produces can handle variations of its teammates in a better way. Moreover, the average score of historical average keeps improving even after the convergence of the best team.

Second, we show an example learning session and the rate of failure of the tested policies. Figure 5.4 illustrates the learning curve for the optimized metric (which is rolling distance for this problem). The second line at the same Figure (Figure 5.4) shows the rate of unfeasible policies that are tried while learning. Unfeasible policies cause failure and return zero fitness as the result. These policies are generally located close to high valued parts of the search space. As an example, consider driving a car as fast as possible on a curve. If you push the limits, the car tips over and it is considered a failure. On the other hand the fastest way to take the curve is close to pushing the limits. What we are looking for is a policy that is fast but not close to failures. For the tensegrity problem, figure 5.4 shows that, the percentage of unfeasible policies that are tested drop to zero. As expected, historical average favors the solutions where slight modifications wouldn't cause failure.

Chapter 4: Tensegrity Locomotion with Historical Average CCEA and Open Loop Signals

Rolling locomotion of tensegrity robots challenges the classical locomotion and controls approaches with the nonlinear interaction of all the elements of a given tensegrity robot. As discussed in Chapter 2, the concept of rolling tensegrity locomotion by body deformation is still an open problem. We address the problem in this section, using open loop controls and CCEAs with historical average.

A typical locomotion research takes a hardware (i.e. a biped robot) and develops a locomotion algorithm for that specific model. As opposed to this approach, in our tensegrity problem, the research on locomotion and the design of the robot is blended in. A physical tensegrity robot that can perform rolling locomotion is also an open problem and the design of such a robot depends on open research on rolling locomotion. Since the rolling algorithm did not exist, building a tensegrity robot that can roll was dependent on multiple questions on required hardware. In this section, we not only provide a rolling algorithm, we also assist the design of the SUPERBall with required hardware such as motor torques, tensions, frequencies, communication requirements etc.

The first step to show feasibility of rolling locomotion is to develop open loop controllers. Open loop controllers do not use any feedback during the control. The motors are controlled with a pregenerated signals that does not depend on the environmental conditions. Although it is a known fact that open loop controllers are not robust to unexpected conditions, they are a perfect match to show feasibility of such a locomotion. Moreover, once we achieve the desired locomotion, we analyze it further to discover necessary components for the design of the SUPERBall.

The rest of the chapter is organized as follows: Section 4.1 discusses the challenges of controlling a tensegrity robot and explains the way we design the open loop controllers for this problem. Section 4.2 explains the usage of CCEA Historical Average to provide rolling locomotion. Section 4.3 investigates the effect different signal types on resulting learned locomotion. We analyze the learned locomotion in Section 4.4 and the role of different muscles in Section 4.4.

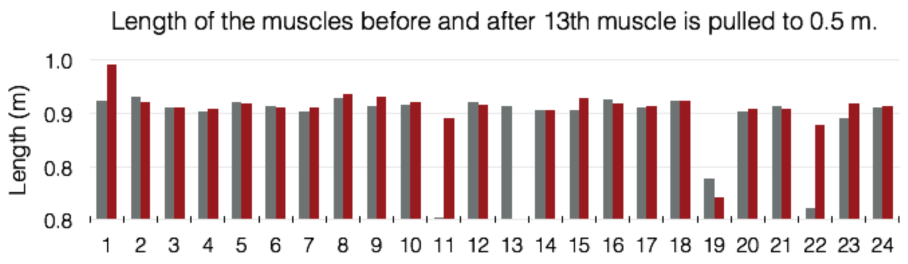


Figure 4.1: Change in length of the muscles, when one of them (the 13th) is pulled to 0.5 meters while other muscles keep the same rest length as before. Grey bars show the original length and red show the final length. While the robot is at the exact same orientation, the actual lengths of the muscles change in a non-linear way. Some of the muscles shorten due to the tension introduced by muscle 13, and some of the muscles relax.

4.1 Controls of a Tensegrity

Controlling a tensegrity robot brings multiple challenges, such as distributed controls, nonlinear interactions between components, and handling difficult to model dynamics, such as oscillations. For this reason, traditional centralized controllers and centralized designs are not a good match for a tensegrity robot. In contrast, we present a decidedly distributed approach for controlling a tensegrity robot. On the hardware side, the core of our design is an independently controllable rod containing two independent "end-cap" controllers on each side of the rod. This model naturally matches the distributed yet holistic nature of a tensegrity. The controllers act independently of each other but interact through the system, where changing the lengths of one of the muscles affects the whole structure in a non-linear way. This behavior can be seen in Figure 4.1. When we pull only one of the muscles (muscle 13), all the muscles change their length, while some of them get shorter and some longer.

To facilitate distributed assembly, the controllers communicate via a wifi wireless network. This design allows for simplified construction and reduces cabling problems that could arise when the tensegrity robot needs to roll through adverse conditions. This design is not only distributed, but modular. For a simple six-bar tensegrity, we simply assemble six identical rods to form the tensegrity robot. In addition, for more complex designs, additional rods can be used without changing the design of the rods themselves.

Our next challenge is to control a set of assembled rods into a high-performance tensegrity robot. To do this, we need controls that are able to work in a distributed control environment, and also work when wireless communication may not be high-bandwidth or reliable. To overcome this problem, we use distributed controls and distributed learning, where each controller learns its policy, but the overall behavior requires coordination of these controllers to make the tensegrity robot move. The setup described is a coordination learning problem where we have independent learners working towards a shared goal. The details of the learning distributed controls for this setup are described in Section 4.2.

An additional control challenge is how to handle the physical hardware limitations of each actuation system. Ideally, we would like our controller to be able to simply dictate the actual lengths of each muscle it is responsible for. However, due to the overall tension caused by the rest of the structure, the controllers can only provide the rest lengths of the muscles. Since the muscles are flexible, the controller changes the actual lengths.

In addition, hardware limitations also play an important role when tensions get higher. Since all the motors in the robot pull against each other, it is possible to reach tensions that the motors cannot handle. Moreover, since the rest of the structure can potentially overpower any one motor, there is a chance that a motor is back-driven and forced to feed out some of the cable stored on the spool. To address these limitations, if a muscle is experiencing tensions above the motor limit, and the cable is pulled to its maximum length of 1.1 meters, the simulation is stopped and the policy is considered infeasible.

To stay within the bounds of the physical hardware, we simulate the motors with high-level controllers that have a constant speed (0.2 m/s) while the tensions stay within reasonable limits. Indeed, the physical motors on the SUPERBall can pull at a rate of 0.5 m/s within the tension range that we are dealing with, but we selected 0.2 m/s in order to leave plenty of hardware headroom, and also to lower power consumption. While the motors move with constant velocity, the controllers dictate preferred positions for the motors. Dictating preferred position is exactly the same as dictating preferred rest length if the cables do not slip. Every timestep, the motors pull or release their cables with a constant speed to get closer to their goal. While staying within reasonable tensions, this setup is feasible on the real robot. The assumption is that there is an intermediate controller layer that regulates the voltage vs. torque in order to provide a constant rotation speed.

The overall goal of the controller is to have the tensegrity robot roll smoothly within the limitations of the actuation and communications hardware. To accomplish this, we use a periodic open-loop controller with parameters that are set by an evolutionary algorithm (note that we have also performed research on closed-loop systems, but due to sensor feedback difficulties and overall increased complexity, we are focusing on open-loop controllers in this paper). During rolling locomotion, the robot (and the controllers) will repeat the same motion (one full revolution) over and over. Considering that the rolling locomotion is a repetitive behavior, signals produced by the controllers will be periodic. The key to making this system work is determining the shape of this periodic signal.

Let's assume that the periodicity of the signal is t and we represent the signal as $F(x)$, x being time within the interval $[0, t]$. There are many possible ways to represent this control function. For instance, a natural choice would be a sine wave, or a series of overlapping waves to form more complex control policies. To reduce complexity, in this paper we use an even simpler control model: we break down each control interval into sub-intervals and assign different preferred rest lengths for each sub-interval. Considering the limited velocity of the motors, the motor will slowly move to reach these selected points during the sub-intervals. The control model is essentially a set of overlapping square waves. As an example, we can divide one period to two sub-intervals, where the motor will have a preferred length of $y1$ for the first half of the signal, and $y2$ for the second half of the signal. With the motor moving towards $y1$ and $y2$, the resulting signal will be similar to the Figure 4.2.

To generate a signal, the only parameters needed are the number of sub-intervals and the rest length values for each sub-interval. For the specific example given in Figure 4.2, the number of sub-intervals is two, and y1 and y2 are the values of the preferred rest lengths for those intervals. The example given in the Figure 4.2 is a simple signal, and while the number of sub-intervals is low, the complexity of signals that can be generated is limited. On the other hand, the complexity of possible signals increases with the number of sub-intervals. Due to this, from now on, we will refer to this parameter as the complexity degree(n)of the signal. Depending on the complexity degree and the values of $y_1, y_2, ...y_n$, the shape of the signal can change between a typical trapezoid, zigzags, stairs or combination of those.

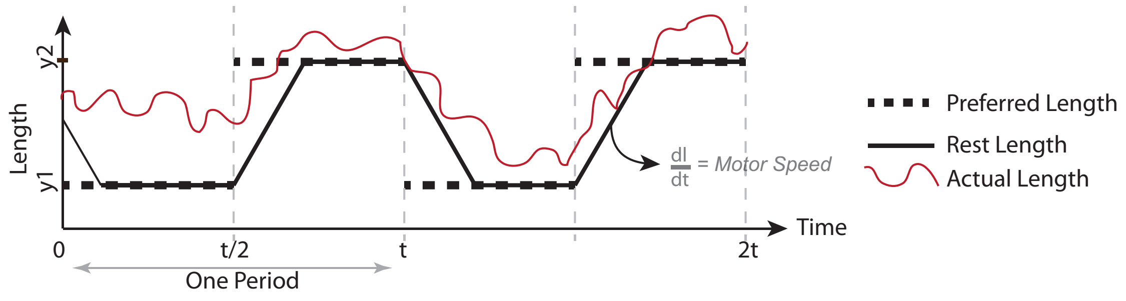


Figure 4.2: An example signal with two sub-intervals with preferred lengths of y1 and y2 and periodicity t.

To summarize, the complexity of the signal depends on n, and each controller has n number of inputs depending on the complexity selected. The rest lengths of the signal follow this signal, and the actual lengths of the muscles change according to the activities of other muscles and the interaction of the robot with the environment. Each controller has a separate signal and it controls only one of the motors. Twenty-four motors control twenty-four muscles independently, but all affect each other to achieve the common goal of rolling locomotion.

4.2 Learning to Roll

While the control parameters to generate the signal are straightforward, the interaction between these signals to reach a rolling behavior is highly complex. As explained before, the nonlinear and oscilattory nature of the problem makes the tensegrity hard to control with classical control methods. The consequences of specific signal combinations can be simulated, but finding the correct signal parameters for a specific behavior is not possible. In this section we explore how we can address this problem by using the simulation combined with a fitness evaluation. As a result, we will implement an evolutionary algorithm that can evolve a set of control parameters that will lead to the desired behavior.

For this study, we used the historical average method that we previously developed and tested with earlier versions of NTRT to obtain rolling behavior using sine wave signals [32]. With the historical average, each member receives its fitness number according to the average of their performances; moreover, if a member survives for the next gen-

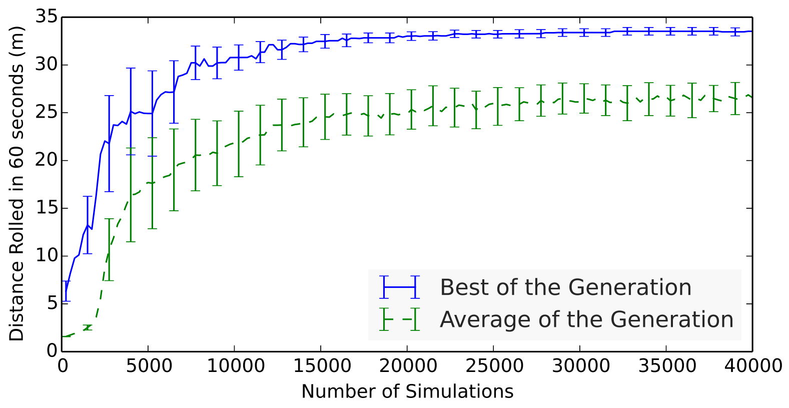


Figure 4.3: Learning curve for the open-loop locomotion with signals with complexity of five degrees and a period of four seconds. The robot learns to roll 30m per minute in 10000 simulations. The error bars disappear, meaning that all statistical runs converge to the same rolling behavior. When the policies are teamed up with other candidates in the population, the robot can still roll 20m in average.

eration (and is not eliminated or mutated), the member keeps its previous experiences. At each generation, the fitness assignment is the average of this growing history of past evaluations. The overall cooperative coevolutionary algorithm with a historical average fitness assignment can be found in Algorithm 3.

First, we show an example learning session using signals with a complexity (n) of five and a period (t) of four seconds. Figure 5.4 illustrates the distance rolled by the robots over the course of learning. Starting with 0 meters, the robots converge, rolling over 32 meters in 60 seconds. This result shows that successful learning of rolling locomotion using CCEA is possible. In Section 4.1, we discussed when a policy is labeled as 'not feasible' during learning. The second line at the same Figure (Figure 5.4) shows the rate of unfeasible policies that are tried while learning to roll. While converging to rolling locomotion, unfeasible policies drop to zero. This shows that the learned policy lies within reasonable lengths and tensions; moreover, it is also far from the limits since small mutations tried during evolution are also feasible.

Considering that the robot has a shape that is similar to a sphere with a diameter of 1.5 meters, rolling 32 meters means approximately seven revolutions in a minute. This results in eight seconds per revolution. Considering that we selected four seconds as the

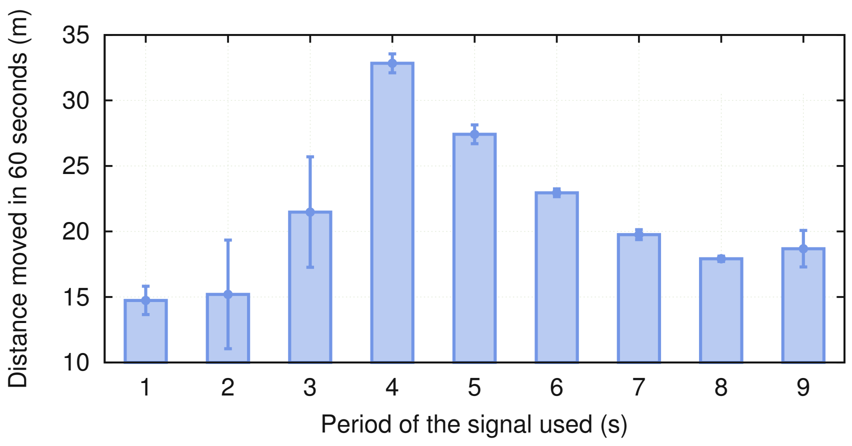


Figure 4.4: The performance of the converged policies after learning for signals with periods of different lengths, while the complexity is fixed to five points. The best performance is reached with signals that are repeated every four seconds. Signals with longer periods have a decreasing performance proportional to the inverse of the periodicity.

periodicity of the signal, the learned signals provide a half revolution, and applying the same signals also results in the other half of the one complete revolution. This supports the reasoning behind selecting periodic signals to obtain rolling locomotion as a periodic movement of the robot.

4.3 Signal Types vs Locomotion

The first set of experiments illustrated when n and t are selected as four and five (four seconds with a complexity of five). Next, we investigate the results of learning using signals with a different complexity and periodicity. Figure 4.4 shows the converged behaviors when we fix n to five and learn using variable t. Note that the signal used for different values of t is not the same. The signals are learned from scratch for each value of t.

Clearly, the peak is when the signals have periods of four seconds (a frequency of 0.25 Hz). When we shorten the period below four seconds, the robot cannot learn to roll. One can think that providing the same signal with a higher frequency can provide the same rolling behavior, but when the tensegrity deforms to start rolling with a higher speed, the contact forces from the ground and the reaction of the structure change completely.

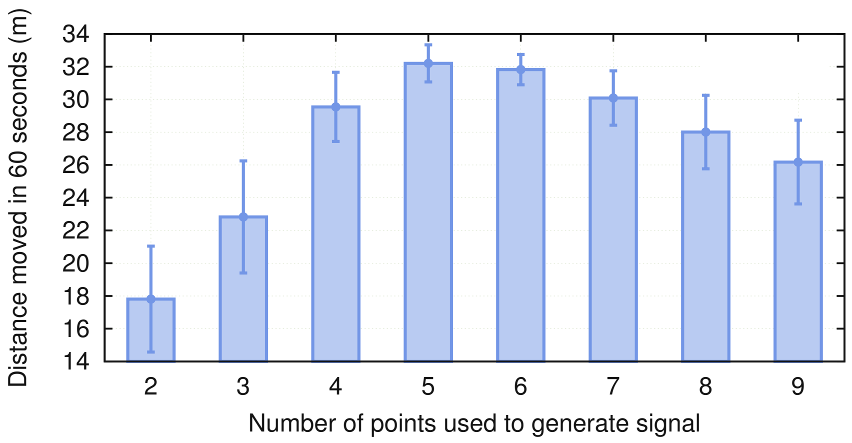


Figure 4.5: The performance of the converged policies after learning for signals with different complexity levels, while the periodicity is fixed to four seconds. The best performance is reached with signals that use five points. Less complex signals cannot generate rolling locomotion, and more complex signals are hard to learn.

When we increase the periodicity to longer than four seconds, the frequency drops and the performance gradually decreases as expected. Moreover, the rolled distance is linearly proportional to the frequency. For the values of 4 to 8 seconds, the performance divided by the frequency gives the same value ($\frac{33}{1/4} \simeq \frac{27}{1/5} \simeq \frac{22.5}{1/6} \simeq \frac{19}{1/7} \simeq 132$).

Next, we investigate the effects of a different signal complexity level to the learning. The period is fixed at four seconds, because it gave the best score combined with the complexity of five in the previous set of experiments (Figure 4.4). We started with a complexity of two, where the signal is as simple as possible. The signal alters between one high value for the first two seconds, and one low value for the last two seconds. We increase the complexity up to nine points. The result is illustrated in Figure 4.5. The first conclusion is that signals with a complexity of two cannot succeed in learning to roll, but the performance increases with higher complexity. Clearly the controllers need more complex signals to provide rolling locomotion. The peak performance is reached at a complexity of five, where the preferred length alters between five different points during five intervals of 0.8 seconds each.

The second conclusion of this experiment is seen when the complexity is increased even further. The learned behavior gradually decreases and error bars show that statistical significance goes down. The reason behind this behavior is that the parameters to learn increase linearly, and the problem to learn becomes linearly harder for each controller. Since all the controllers learn simultaneously while interacting with each other, overall difficulty of the problem is increased even further. The error bars show that in more complex problems, some statistical tests achieve good results while some of them fail completely due to the difficulty of the problem.

4.4 Analyzing the Rolling Behavior

In previous section, we showed that learning to roll for a tensegrity robot succeeds with signals of right periodicity and complexity. In this section, we look at the learned behavior and analyze the feasibility of the behavior, lengths and tensions during rolling, converged signals, and robustness of the behavior.

As a sample learning behavior, we select one of the learned behaviors with a period of four seconds and a complexity of five. The learning process for this behavior is illustrated in Section 4.2 and Figure 5.4. For each simulation, the robot tests different policies for sixty seconds, and the distance moved is marked as the score. The policies that are tested are updated according to the Cooperative Coevolutionary Algorithm with a historical average (Algorithm 3). For this particular experiment, the robots reach the performance of rolling 32 meters around 5,000 simulation steps. As a side result, the percentage of failed policies (due to high tension) reaches zero.

First, we take this learned behavior and look at the learned policy. Figure 4.6 shows the intervals that each muscle's length lies within. The muscles' lengths vary from 0.7 m to 1.1 m. While some of the muscles, such as numbers 1 and 21, have minimal change in length, some of them, such as numbers 13 and 23, have larger changes and bigger intervals. Another remark is that the mean of the signal (that is noted by the red horizontal line) is not necessarily at the center of the interval that the signal lies in. This is one difference that more complex signals provide. For example, the length of the number 1 muscle is mostly around 0.82 m, but it reaches as low as 0.7 m once in a while.

One way to analyze rolling behavior is by looking at the average lengths and tensions of the muscles in addition to the potential and kinetic energy of the structure. First, we look at how the actual lengths of the muscles change compared with the signals provided. Figure 5.10 shows the average rest length of the muscles (signal provided) and the average actual length of the muscles. The area between the two lines shows the

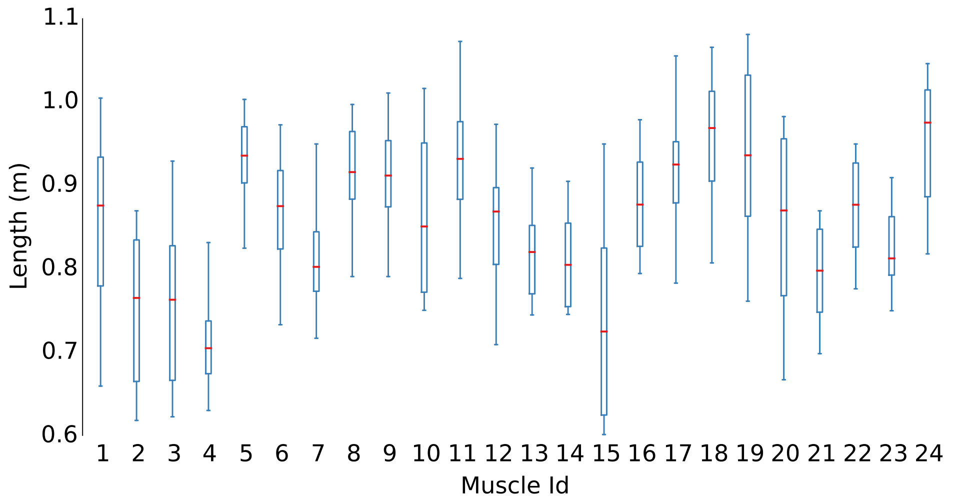
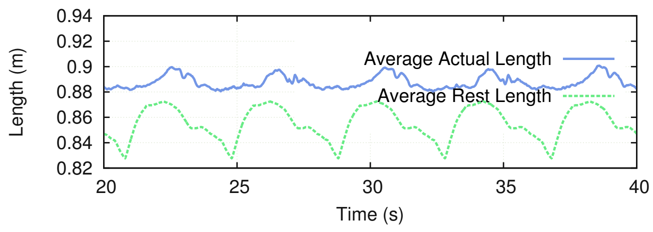


Figure 4.6: A sample learned policy for twenty-four motors is illustrated. For each signal, the red line at the center shows the mean of the signal and the box and dashed lines show the interval that the signal lies in.

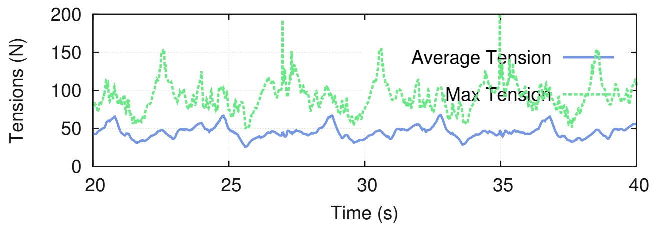
stretch of the muscles due to the tension. The most interesting fact about this graph is the difference between frequencies for the two lines. Although signals that are provided to the muscles repeat themselves every four seconds, actual lengths repeat themselves every eight seconds. This supports our previous conclusion about using signals that have a periodicity of four seconds that can conclude in revolutions that take eight seconds. The first and second halves of the roll use same signal, but ground interactions make the actual lengths differ.

The average and maximum tensions of the muscles during rolling are illustrated in Figure 4.7b. The average tension is low and stays around 60N. The second line shows the the tension of the muscle with the longest stretch at each particular time of the simulation. The value goes up to 200N, staying within values that our hardware design can handle. The maximum tension graph also repeats itself every eight seconds as expected.

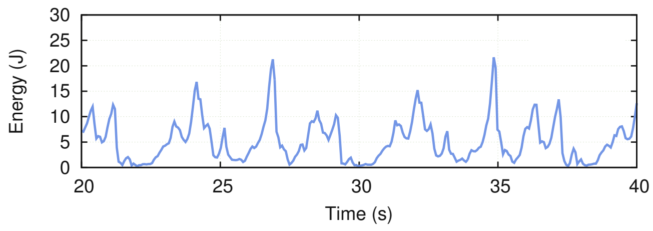
When we observed the gait learned using our simulator, we see that the rolling locomotion does not have a constant speed. Instead, it slows down and speeds up periodically during each revolution. To illustrate this behavior, we looked at the total



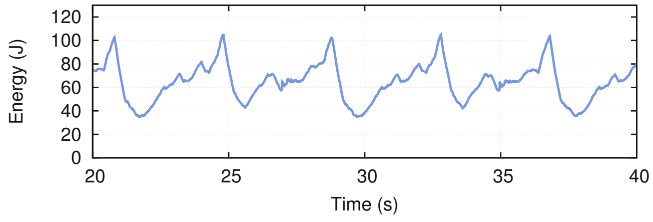
(a) Average Rest Lengths and Actual Lengths of The Muscles



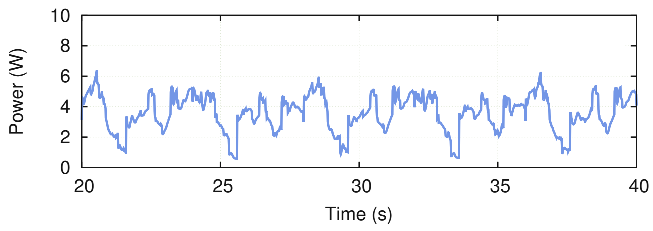
(b) Average and Maximum Tensions of the Muscles



(c) Kinetic Energy of the Tensegrity Robot



(d) Total Potential Energy Stored in Muscles



(e) Power Used by the Motors to Roll

Figure 4.7: Illustration of different aspects of the Tensegrity Robot over time, during rolling locomotion. The used signals for muscles repeat themselves every four seconds. The tensegrity robot completes one revolution in eight seconds. Tensions, lengths, and power usage of the robot stay in our defined hardware limits.

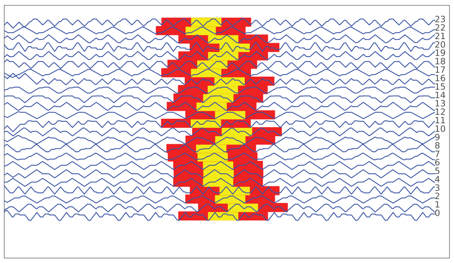
kinetic energy of all the rods over time as seen in Figure 4.7c. If we take the interval between two peak points (when t=27 and t=35), the kinetic energy stays at zero for one second, which is around t=30s. Moreover, the repetitive acceleration and deceleration can clearly be seen. This behavior creates an inefficiency in terms of energy for the gait. There are two main reasons for this behavior: First, the learning algorithm only optimizes the distance rolled, not the energy spent during motion. Second, in this work, we are testing open-loop controllers. Using some feedback from the robot (such as lengths, tensions, or orientation), having a smoother rolling experience can be possible. This problem is addressed in the next Chapter with a rolling algorithm that uses sensor feedback.

Next, we observe the potential energy stored in the muscles. Figure 4.7d shows the pattern that repeats itself every eight seconds as expected. The first and second four seconds are similar, but they differ slightly due to different reactions with the environment during the second half of a complete roll. The pattern shows an overall behavior of increasing the potential energy slowly over time, and releasing it. This matches the kinetic energy behavior that we observed, as seen in Figure 4.7c. The kinetic energy of the structure increases during the few seconds following the moment potential energy is released (i.e., t=25s).

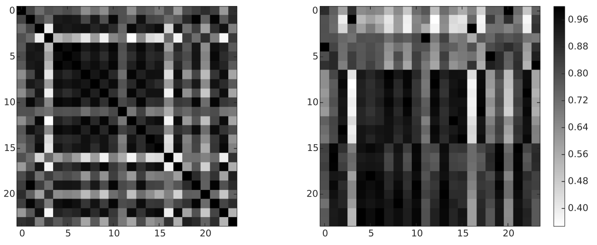
The last set of experiments analyzes the approximated power usage by the motors during rolling locomotion. In simulation, power consumption is approximated using the current tension of the element and the constant speed with which the motors shorten the muscles. As we explained earlier, the learned behavior is not optimized to be power-efficient for this study. On the other hand, we want to make sure that the required power is within the limits of the motors and batteries that will be used in the hardware. Figure 4.7e illustrates the average power consumption of the muscles that varies between 2 to 6 W per muscle. Considering that the motors always pull against the tension (and all the muscles are tight all the time), this value is considerably low. Moreover, it is always possible to lower this value by using a feedback controller in future work.

4.5 Analyzing the Roles of Different Muscles

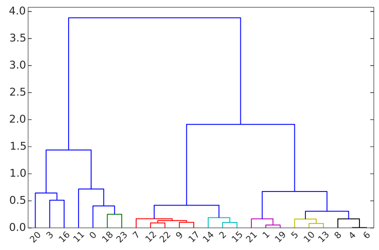
Next, we analyze the signals further by looking at their shapes and the correlation between them. The top half of Figure 4.8a shows all twenty-four signals when they



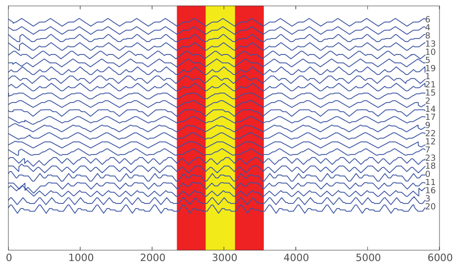
(a) The signals of the twenty-four muscles normalized between [i,i+1] for muscle i. After correlation using time offsets, the highest correlated intervals of four seconds are marked with a red-yellow-red pattern.



(b) 24x24 Correlation matrix of signals used for each muscle before and after reordering using hierarchical clustering.



(c) Result of Hierarchical clustering to group and reorder similar signals



(d) The signals for muscles during shifting according to the highest correlation, with reordering according to hierarchical clustering.

Figure 4.8: The process of analyzing the signals used for the muscles. Signals are shifted and reordered to show similarities. Subfigure (d) shows that groups of signals have similar patterns.

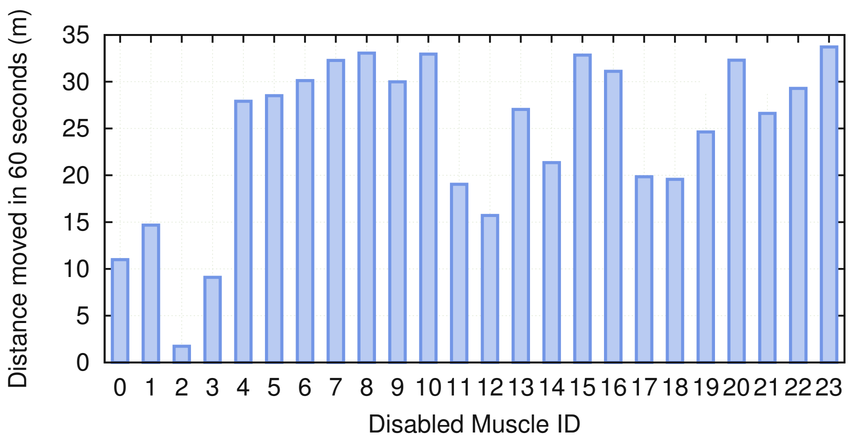


Figure 4.9: The performance of the learned policy when one of the muscles is disabled. Learned policy is partially robust to failures of some muscles.

are normalized between zero and one. The purpose of this experiment is to show the similarities of the signals and different types of signals learned at the end of coevolution. First, we look at the correlation between signals. For each signal, the red-yellow-red area highlights the interval that maximizes correlation with other signals. In this ordering, it is hard to see the similarities between signals. As shown in Figure 4.8d, we shift the signals so that their selected intervals match, then we use hierarchical clustering (Figure 5.13) to group the signals according to the similarity metric.

After reordering the signals, Figure 4.8d shows that half of the signals have one peak high and one peak low, but the other half have two signals that are more complex with two peak points. This result gives us multiple conclusions. First, the learning algorithm makes use of the complexity provided. Although a subset of the signals is simple, another subset has more complex signals with multiple peak points that can only be generated with complexity coefficients that are higher than three. The subsets of the signals that are similar can be regenerated using different parameters, but with the same formula. Let's consider a normalized signal as $f(x)$. The formula $g(x) = A + B * f(x + C)$ can produce similar signals for different values of A, B, and C. Using this idea, all twenty-four signals can be reproduced using three to four base functions and different parameters. This gives us a hint about why many papers in literature propose to use Central Pattern Generators (CPGs) to control tensegrity robots.

The last set of results shows how critical each muscle is for a given rolling locomotion. Taking the tensegrity robot with the learned policy, we disable one of the muscles and observe the effect of such a failure on its overall behavior. Figure 5.20 shows that performance depends on which specific muscle fails. For a significant number of muscles, using the same algorithm still provides rolling behavior with similar performance. Conversely though, some muscles roles are critical to the algorithm's succcess.

4.6 Conclusions

The rolling locomotion for a tensegrity robot has many challenges that causes problem for classical control methods. These challenges limit both the number of studies on tensegrity locomotion as well as development of tensegrity robots. In this scenario, design and development of SUPERBall and research on locomotion are two problems that are dependent on each other. In this chapter, we break this loop by combining open loop controllers and coevolutionary algorithms.

We approach the problem by using distributed controls that matches distributed nature of the tensegrity robots. There are 24 controllers that controls 24 muscles, and each controller uses an open loop signal to change the lengths of the muscles. We investigate the learned behavior using different types of signals with different frequencies. We conclude that a robot with SUPERBall specifications (1.5 m long and 2 kg rods each) can roll 0.5 m/s while keeping tensions and torques within a reasonable range.

Open loop controllers are a simple way to control a tensegrity robot. Since they do not take the environment into consideration, they can not respond to unexpected conditions. Despite its disadvantages, they are a good match for the purpose of analyzing the capabilities of tensegrity robots. Combined with open loop signals, we show that CCEAs have the capability to address this nonlinear problem and optimize the rolling distance.

The main contribution of this chapter is to show the feasibility of rolling locomotion for an icosahedron tensegrity robot. Second, we observe the learned behavior and the trajectory of the robot. The numbers given on tensions and power consumption give an insight for further design of the prototype of the SUPERBall.

Open loop controllers are the first step to rolling locomotion. To be able to truly control a tensegrity robot, the locomotion algorithm has to be steerable, robust and responsive to the environment conditions. Handling external and unexpected conditions requires the usage of sensor information in controls loop. In next chapter, we present a learning based closed loop locomotion algorithm that provides these properties.

Chapter 5: Locomotion using Flop and Roll

We showed the feasibility of rolling locomotion with open loop signals and evolutionary algorithms in Chapter 4. The open loop approach provides a non-directional rolling behavior that is prone to unexpected external forces or terrain conditions. In this chapter, we provide a learning based closed loop controls algorithm that uses the contact sensor information of the robot as the feedback. We first divide the problem into simple flops. Next, the robot learns to make flops that will optimize the rolling behavior. We combine this with policy pooling, a method that we develop to take advantage of the symmetry of the structure. These two methods combined with coevolutionary algorithms provide a learning locomotion that steerable and robust to different environment conditions.

5.1 Flop and Roll

Locomotion using feedback is a complex problem. The first step that we take is to divide the problem of rolling into consecutive flops. Considering that the structure is stable on one of its surfaces, we define 'a flop' as deforming the structure and falling to one side only to end up lying on another surface. Doing one flop towards the target will move the robot towards the flop direction and change its orientation. Following the same routine over and over will end up moving the tensegrity robot in the desired direction. Learning to do a single flop in a desired direction is a simpler problem than learning smooth rolling. Unlike the previous approach of learning to roll, what the robot optimizes is a simple 'flop' behavior with the help of feedback from its sensors.

On the other hand, learning a single flop and repeating it does not necessarily provide a smooth rolling locomotion. As an analogy, the difference is similar to the difference between repeating the routine 'one step forward and stop' and smooth walking. During smooth walking, steps taken are optimized for consecutive steps and they differ from taking one single step forward. To avoid such a difference, we chose our fitness function to evaluate overall rolling. During the evolution of policies, the policy that we evaluate makes a single flop, but these single flops will evolve according to their success when they are executed consecutively over 60 seconds. With the same analogy of walking, we evolve the robot to learn how to make a step, but the policy to perform each step is evolved so that it will maximize the walking behavior when executed over and over. The details of the learning algorithm and the state and the fitness function will be explained in Section 5.3

The first advantage of the approach that we take is making the control policy simpler (single flop), while learning a more complex behavior (smooth rolling). Additionally, the algorithm can handle external and unexpected forces during rolling motion. This robustness is mainly provided by the fact that the algorithm is composed of smaller pieces to make each flop as opposed to previous research that provides the whole rolling sequence on a given surface [32]. Let's imagine a tensegrity robot that encounters a large external force while rolling using "flop and roll" towards a target point. The external force applied will break the sequence of flops for the robot, and the robot will end up in a random orientation. Since the robot has a spherical and symmetrical nature, it will land on one of its faces. The robot then executes the algorithm in the new orientation, and will deform itself to undertake the first flop that will be followed by a rolling behavior towards the desired target point.

Finally, we test the robustness of the algorithm with an environment where executing a flop is harder due to different terrain properties or external forces. As learned, the robot will try to execute the single flop until it lands on a different surface. The algorithm considers itself at the same state and works on completing the flop until it succeeds. This property allows the algorithm to work on uneven terrains, hills, small obstacles, and unexpected external forces as shown in the results in Sections 5.3.4 and 5.3.5.

5.2 Distributed Controls via Pooling

We first discuss how to control the robot using distributed controls while still taking advantage of the symmetrical nature of the structure. The ideal control algorithm for the robot provides rolling motion that is steerable in a desired direction and robust to external forces. This is the main significance of the rolling algorithm that we present in our paper. In terms of input, the algorithm takes sensor information from the robot. Traditionally, robots are equipped with many different sensors, such as cameras and infrared sensors. We will use the minimal information, such as pressure or contact

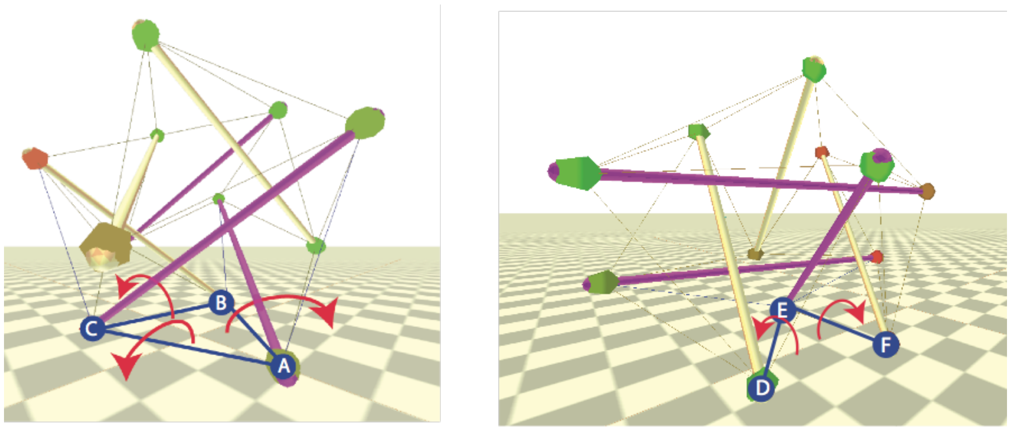


Figure 5.1: An icosahedron tensegrity robot has only two possible configurations when it is balanced. In the case on the left, the robot is lying on an equilateral triangle composed of nodes A,B, and C. There are three possibilities for the next flop marked with red arrows (AB, BC, or CA). On the right, the robot is lying on an isosceles triangle with nodes D,E, and F. There is not any muscle connecting D to F. There are two possibilities for the next flop (DE and EF). We do not consider the flop over DF since it requires lot more deformation.

sensors, at the end of the rods and the desired direction.

The robot has 24 muscles that are controlled by 24 independent controllers. Each controller is responsible for selecting the desired length for the muscle that they control. One important point here is that the algorithm does not directly control the lengths of the muscles at a given time. Since the algorithm is based on simple flops for the robot, the lengths provided are the desired lengths for the muscles, so that when reached to the configuration, the structure will flop and start rolling. This particular point makes the algorithm time-independent.

The actual lengths of the muscles at a particular time depends on the speed of the motors and previous configuration, but the algorithm still works with slower motors since reaching the desired configuration will make the tensegrity flop. This fact is also supported with experimental results in Section 5.3.4. Another advantage of time-independency is that the controllers do not need to have precise synchronization. The robot can tolerate latency in coordination and will still perform the flop.

At each moment, the robot has a state (contact points and goal direction), the controllers (24 total) and actions to choose (preferred lengths). The goal of the algorithm is to provide rolling behavior that is composed of single flops. The set of policies are defined as

$$\pi_x : (\phi, \upsilon) \rightarrow (l_x) | x \in \{1, 2, .., 24\}$$

where ϕ is the contact points and υ is the desired direction, and l_x is the desired length for the muscle x. Now we will reduce the complexity of the policies to learn by what we call 'pooling', which takes advantage of the symmetrical nature-repetitive pattern of the tensegrity structures.

Let's first analyze the stable configurations for the robot in its default configuration (equal lengths for all the muscles). When the structure will be stable on a surface, it will have 3 points of contact forming the base triangle. Here, we use this advantage for discretization of the state space. Instead of having the orientation of the structure, we use the base triangle that the structure is lying on. For the desired direction, we use the sides of the base triangle, which gives us 3 possible values. We define the new state variables as:

$$s : (\overline{XYZ}, d),$$

where X, Y, and Z represent the edges of the base triangle and d takes values of 0, 1 or 2 (XY, YZ, or ZX), representing the side of the triangle that encapsulates the desired direction to roll.

There are twenty possible surfaces for the icosahedron robot. Since tensegrity robots have repetitive patterns, there are only two types of base triangles: an equilateral triangle where all 3 nodes are connected, or an isosceles triangle where only two of the sides are connected. Figure 5.1 shows the only two types of possible base configurations. Out of twenty triangle surfaces, eight of them are equilateral (Figure 5.1 - left) and twelve of them are isocele triangles (Figure 5.1 - right). At any one of these stable situations, the goal of the controllers is to make the robot flop on one of the sides of the triangle. The possibilities are sides AB, BC, and CA on the left of Figure 5.1 and DE and EF on the right of Figure 5.1. We do not consider flopping over DF, because not only is DF not connected, it is impossible to perform that flop with a small deformation, since the projection of the center of mass of the structure is much closer to point E, as opposed to the edge DF.

Let's assume that all the controllers have the policy to flop in a given state of

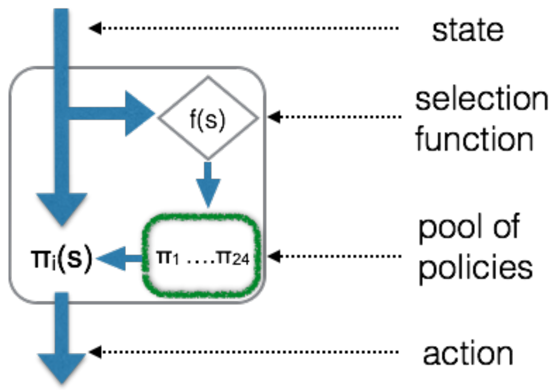


Figure 5.2: Overview of the agents in flop and roll algorithm. The pool has 24 different policies. The agent first receives the state and selection function decides on which policy to use. The selected policy decides on the action given the state.

$\overline{ABC}, AB$. It can be seen that the structure is symmetrical for all 3 sides (Rotating 120 degrees around a gravitational axis will give the exact same structure). Moreover, if the structure is lying on any other equilateral triangles, we will see the same pattern of connections. This resemblance lets us reuse the knowledge of the policies for state $\overline{ABC}, AB$ for every equilateral base, and $\overline{DEF}, DE$ for every isosceles triangle. The idea is similar to transfer learning [78]. We do not directly copy the knowledge, instead, the policies that perform these flops are reused for flops in all possible orientations.

To reuse the knowledge gathered, we use a virtual pool of policies, assuming that all the learned policies $(\pi_1, .., \pi_{24})$ for those particular states ($\overline{ABC}, AB$ and $\overline{DEF}, DE$) are available to all of the controllers. In a new state s' , we only need a function F that returns the policy that each controller select from the pool so that the desired flop will happen.

$$F : (s, i) \rightarrow (j_\pi),$$

where s is the state, i is the unique ID of the controller, and j_π is the ID of the policy that the the controller i should pick from the pool. Figure 5.2 illustrates the internal structure of the agents. The bigger picture has coevolution of pool of policies. We illustrated this flow in Figure 5.3. Coevolutionary algorithms gives a sample team to try. This sample team forms the pool of 24 policies. This pool is copied to all of the agents. The agents are used for each of the muscles to perform rolling. According to the resulting behavior,

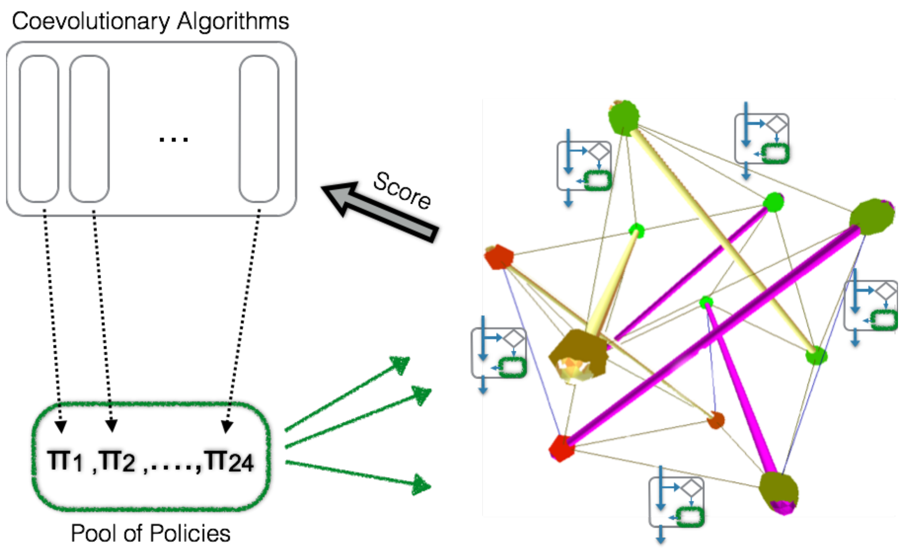


Figure 5.3: Overview of the pooling in flop and roll algorithm. Coevolutionary algorithm coevolves 24 different populations policies. One policy per population is selected to form a sample pool. The selected pool is embedded to the 24 agents controlling 24 muscles. The simulation returns a score that is used for fitness of the individuals.

the simulation returns back a score for the used pool. This score is returned back to the coevolutionary algorithms. In coevolutionary algorithms, this score is used to assign fitness to the selected policies. Please note that the given method is not specific to our robot, F can easily be designed or discovered using the repetitive nature of tensegrity structures. Using pooling, we now reduced the complexity of the problem to learn to 24 policies with only two different states. 24 policies combined with the pooling function F will provide the capability to flop in every possible orientation.

An alternative explanation to this pooling mechanism uses 'roles.' Depending on the new state, each controller selects one of the 24 roles. For example, the controllers of the base muscles in current condition will select the roles of the muscles AB, BC, and CD. The policies in the pool actually represent what to do for that specific role to make a flop. This selection function F , is hand-coded according to the structure, but the policies for the roles are learned using coeovolutionary algorithms.

Sharing a pool of policies gives the impression of excessive communication. Yet, there is no active communication during rolling. The policies to use are decided before each episode, and they are not updated during each trial of rolling. Since we use evolutionary algorithms, and delayed fitness assignment, the policies are only updated before each episode. The only time that controllers have to communicate is before starting an experiment, to make sure that their pool is synchronized. Once it is synchronized, the robot can start the experiment and roll without the need of communication for policies. A small amount of communication is used to make sure all the agents figure out and learn the current state. The state is basically a binary value of a pressure sensor for each node (twelve bits total), and this communication is not sensitive to timing as discussed earlier. It is also possible to derive the state without communication (i.e., using tension sensors, proximity sensors, etc.), but we leave that for future research.

5.3 Learning to Roll via Flops

Pooling (Section 5.2) reduced the problem to learn to flop with 24 agents in two possible states. On the other hand, the higher-level function to learn is to roll. We will learn the pool of policies, where each policy controls one muscle of the robot for a specific orientation. As described in Chapter 3, we use coevolutionary algorithms and historical average reward shaping for learning these policies.

Same as the open loop experiments in Chapter 4, the problem is episodic. The agents have 60 seconds to test the policies in the pool, then we reset the experiment. At the start of the episode all the agents copy the pool to minimize communication during rolling. At each state s, each agent calls the function $F(s, i)$ to select the policy π_j from the pool. Then agents use the policy until the state changes to a different state, meaning that the structure performed one flop and ended up on another surface.

At the end of each episode, all of the policies forming that team are evaluated according to the performance of the whole robot after 60 seconds. The global fitness function is the distance covered towards the desired goal point. Once again, fitness is not related to the flops, it is only related to the overall rolling.

Although it is possible to evaluate all these policies with the same fitness function (distance traveled), a better fitness assignment can evaluate each policy according to their contribution to the overall performance. As described in Chapters 2 and 3, fitness shaping can provide better learning for cooperative learning. Moreover, since we want a more robust solution, we use historical average fitness shaping that we defined in

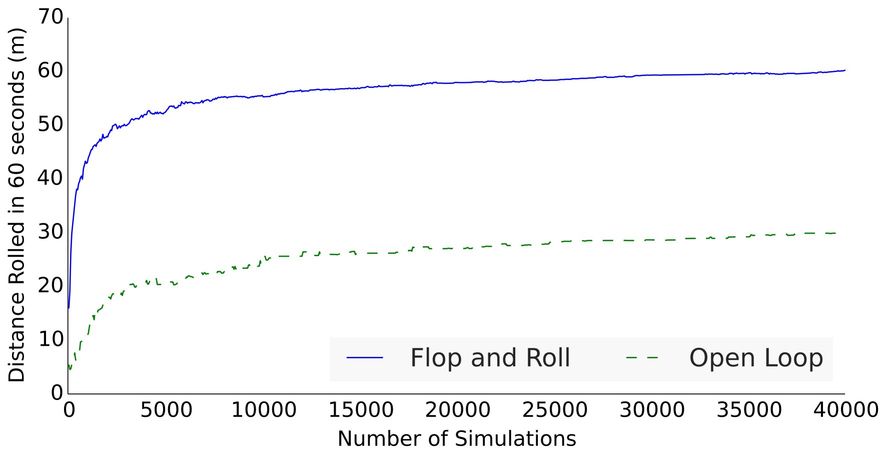


Figure 5.4: The performance of the best policies over time during the learning process. Flop and roll is compared to the non-directional sine wave approach. Flop and roll learns directional rolling behavior in a short amount of time.

Chapter 3 [30, 31]. In the historical average, each policy in the pool shapes the global fitness according to its previous experience by taking the average of all the teams that it has been previously tested with. As we have seen in Chapter 4, the rolling behavior that is learned using historical average is less affected by small changes in the environment. To show its robustness, we will perform different tests with the learned behavior in later sections of this chapter.

For each generation, we test the individuals using their performances with different teams. To form teams we use random sampling. 50 random teams are formed and the members are assigned fitness according to these experiments. After 50 experiments, each population eliminates half of its members to keep the most fit ones. These selected members form new members by mutation.

We now present the results of the experiments that we conduct using NTRT. In our experiments, each strut is 1.5m in length, 3 kg in weight, and the same as in the specifications of the SUPERball design in Chapter 8. We used 24 active muscles controlled by 24 controllers, where the muscles have a rate of change in length of 0.3 m/s, and the elasticity coefficient for the muscles is 3kN/m.

The first experiment is to learn to roll using the algorithm ‘flop and roll,’ as described in previous sections. We compare it to the sine wave rolling algorithm that was previously presented in Chapter 4. First, flop and roll is a method for learning to roll towards a goal; on the other hand, the sine wave approach that we compare it to is a less-capable algorithm that provides uncontrollable rolling motion in one direction. To be fair on the sine wave approach, we compare the distance traveled in any direction by the sine wave approach vs. the distance traveled towards the goal using flop and roll.

Figure 5.4 shows that the flop and roll algorithm actually takes less time to learn and perform better. The flop and roll algorithm takes advantage of the reuse of knowledge and has a learning curve with a jumpstart. Flop and roll learns a basic rolling behavior in a few generations, and it reaches rolling with a speed of 60 meters per minute. In addition to being controllable and direction based, flop and roll is faster to learn than open loop controls.

In this figure, the result of the open loop approach is slightly different than the original result that was published in 2013 [32]. Since the first publication of the open loop approach, the realism of the simulation environment is increased using improvements that are made on NTRT. Moreover, in this work, the specifications of the tensegrity model is closer to the prototype that is currently under development [10]. As a summary, here we tested the two algorithms in a more realistic environment with a robot that is harder to control.

5.3.1 Rolling locomotion on a straight path

Second, we look at the learned rolling behavior in details. We take the best policy learned and test it with a stationary target. Figure 5.5 shows the projection of the center of mass of the robot during the 60 seconds time window. This particular pool of policies rolls 80 meters on that given direction. The zig-zag behavior that is observed is due to the fact that the robot does not have a perfect spherical shape. As expected, icosahedron robot has to roll on triangle surfaces instead of a straight path.

The trajectory of the robot is slightly angled during first few meters. First, the steering that is seen in the picture is slightly exaggerated due to the difference in scales of two axes. The scales are different to show the zigzag pattern as well as the general trajectory. The initial steering is mainly caused by the difference between starting the locomotion and keeping to roll. The only feedback that the robot uses is the base triangle.

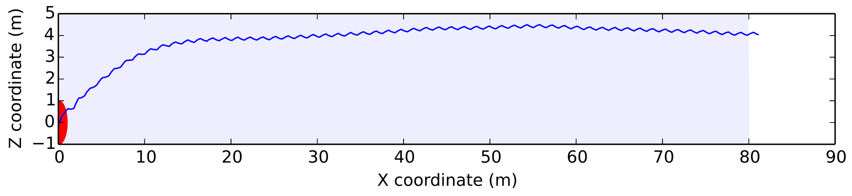


Figure 5.5: The 2D path followed by the robot using towards a stationary target.

The algorithm does not take into consideration the speed of the robot. The robot uses exact same actions both to keep rolling and to start rolling.

Learning graph (Figure 5.4) showed us 60 meters of average rolling per minute, Figure 5.5 shows a trajectory of 80 meters for the same time window. The reason behind this is the location of the target point. That particular angle is one of the natural rolling directions for the icosahedron robot. Since the robot is icosahedron, and algorithm is based on triangle surfaces, there are 3 different natural rolling directions. To illustrate these directions, we place the target to different locations and illustrate the distances rolled and trajectories followed by the robot.

Figure 5.6 shows the performance of rolling to different directions. At each experiment, we place the stationary target a different direction and measure the distance between the final position and starting position. As expected, there are 3 major directions where the robot rolls 80 meters. In weakest directions, the total distance covered is around 55 meters per minute. The main reason behind this is the path followed by the robot while traveling to these weak directions.

Figure 5.7 shows the trajectories of the robot during the same experiment. It can be seen that the robot can use a straight path to these 3 natural directions. On the other hand, when the target is in between these 3 directions, the robot cannot directly roll towards the target. First, it starts rolling in the natural rolling direction that is closest to target direction. When another rolling direction (or another side of the base triangle) provides a closer rolling direction, the robot makes a sharp turn. Overall the rolled distance by the robot is still close to 80 meters, but the distance between the starting and the ending points is limited to 55 meters due to triangle bases of the icosahedron shape of the robot.

The reason behind these sharp turns is simple. While designing the flop and roll

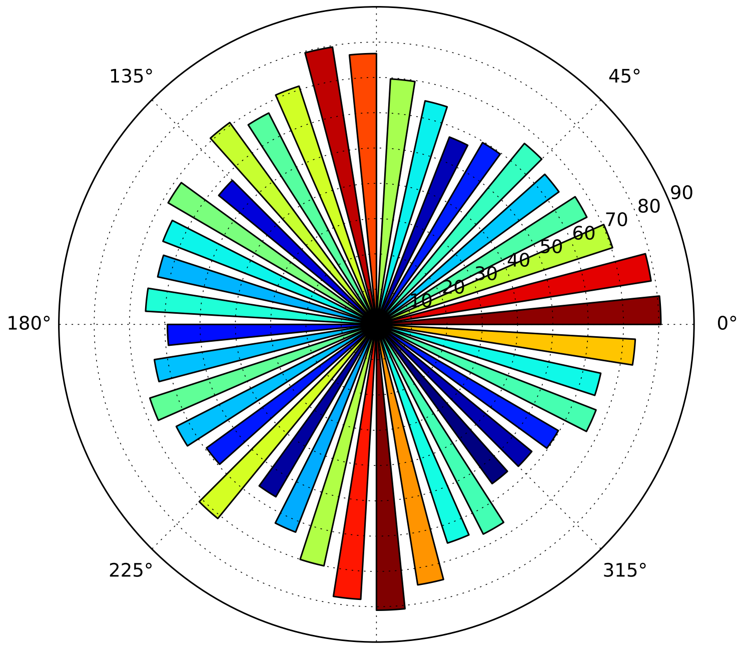


Figure 5.6: The distances that the robot rolls when the target is placed in different directions. 3 natural rolling directions provides fastest rolling experience.

algorithm, we selected desired direction as an input. This desired input is indirectly discretized, because the algorithm performs the flop the closest side of the base triangle. The behavior is optimized for continuous rolling on a straight path. Considering these aspects of the algorithm, steering the locomotion is not possible. The robot performs rolling locomotion on a straight path and then does a sharp turn when necessary.

5.3.2 Analysis of the Locomotion

In this section, we are analyzing the robot's attributes during locomotion. We are going to illustrate and explain the shape of the structure, the trajectory of the robot, the kinetic and potential energy during the course of locomotion. This analysis will bring the high level emerged behavior used by the robot during the course of locomotion and it will also give an idea about further strategies to develop different rolling locomotion algorithms.

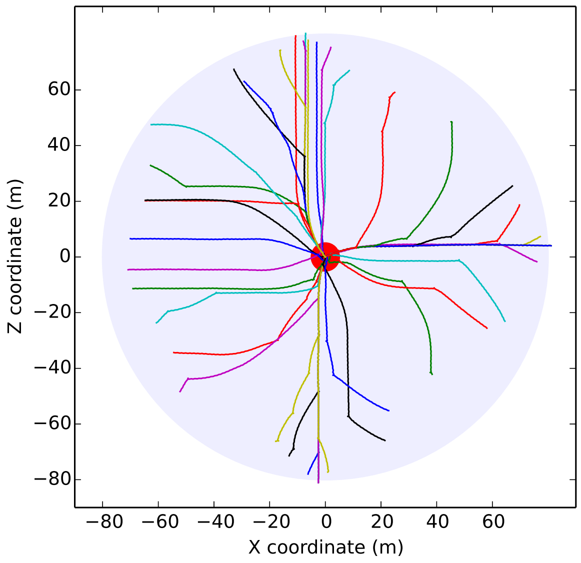
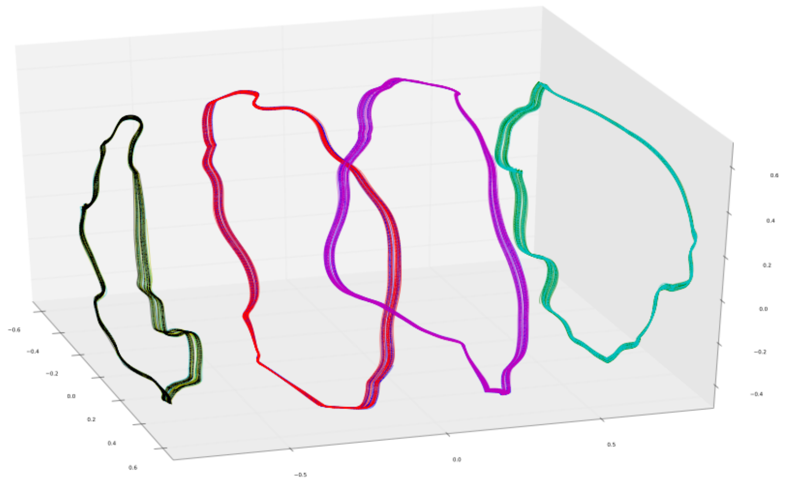


Figure 5.7: The trajectory of the robot when the target is placed in different directions. 3 natural rolling directions provides straight path.

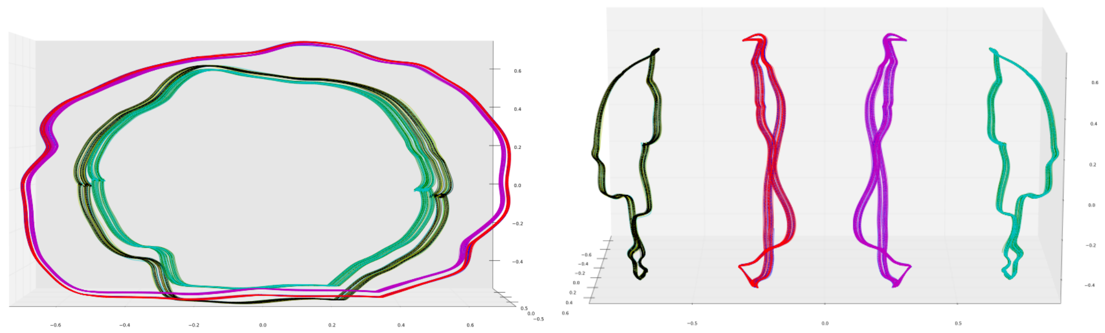
During the design of the flop and roll algorithm, we exploited the symmetry of the structure. As expected, patterns arise in the learned behavior too. First, we look at the shape of the structure. One common way to illustrate a legged locomotion is to look at the paths followed by the legs compared to the center of the gravity. Figure 7.4 shows these trajectories from 3 different angles.

The most important aspect of this figure is the ellipse figure created by the robot. During the locomotion, the robot deforms its overall sphere to be slightly compressed from top due to gravity, and slightly expanded towards the sides due to rolling direction. This resulting deformation gives to the robot a good balance sideways because of the contacting area with the ground. In addition, the contact area that is shorter on the rolling direction makes it easy for the robot to tip over towards that side.

In addition to the overall shape there are more conclusion to be drawn from this figure. The robot has 12 end caps and all the end caps contact the ground. We can compare these end caps to 12 different legs that are interchangeably used depending on



(a) view from 45 degree



(b) view from the side

(c) view from the back

Figure 5.8: The trajectories of the end of the rods (12 total) with respect to the center of mass of the robot during 60 seconds of locomotion.

the orientation. The trajectories of these 12 legs results in the following conclusions: follow 4 main trajectories. For each side, there are 2 main classes: outer legs and inner legs. The results that we conclude from Figure 7.4 are follows:

- There are 4 classes and 3 legs in each class.
- For each side, there are 2 main classes: outer legs and inner legs.
- The legs that are at the same class follow the exact same trajectory.
- Inner end caps stay on the ground for a longer period of time, while outer legs stay longer.

From the design of our algorithm, and the icosahedron shape of the robot , it is a known fact that the contact area (base) at a specific time instance is a triangle. While looking at the trajectory of the center of mass (Figure 5.5), zigzag pattern emerges. Combining these two facts with the symmetry in Figure 7.4, it is possible to discover the pattern of base triangles during the locomotion.

On a sphere robot, the most trivial way to move in one direction would be moving center of mass towards the desired direction. As opposed to our intuition, the learned behavior establishes forward locomotion by moving the center of mass sideways to the weaker side that will establish a diagonal flop. Overall strategy used by the robot is based on moving the center of mass to one side, while being on triangle base located on on the opposite side. This strategy can also be observed in biped locomotion. Moreover, the patterns of the base triangles also supports this analogy with biped locomotion.

Next, we observe the kinetic and potential energy of the structure over the course of first 20 seconds. The goal here is to show that:

- the robot does a continuous rolling locomotion instead of series of flops.
- the overall stiffness stays the same during different phases of locomotion.

Figure 5.9 shows that the first few flops are not completely connected. The kinetic energy starts from zero goes up and returns back to zero couple of times. After several flops, the rest of the locomotion shows a pattern without any complete stops. The small variance is caused by the end of the rods hitting the ground after each flop, but the structure keeps rolling despite the impact. The potential energy shows that the structure loosens

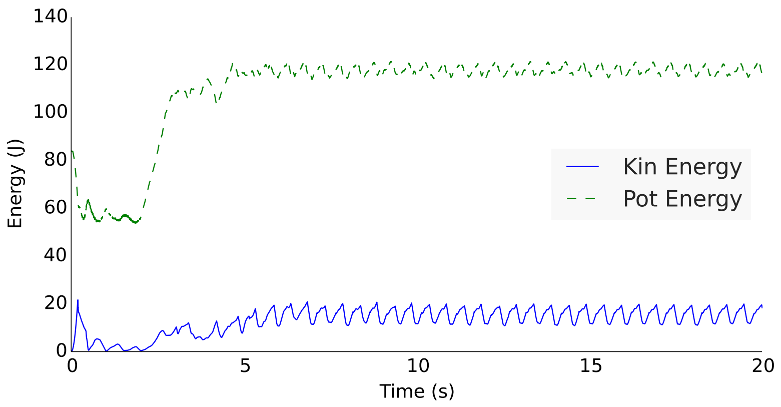


Figure 5.9: The kinetic and the potential energy of the robot.

itself during the first flop, but the overall stiffness is higher during the locomotion than it is at start. Higher stiffness after first few seconds is not directly caused by the muscle activity. It is a consequence of the compliance of the structure and the interaction with the ground during the locomotion.

5.3.3 Analysis of the muscles

In this section, we analyze the active components that drives the robot to provide the rolling locomotion. Average and maximum length and tensions of the muscles both shows the feasibility of the algorithm on a physical robot and gives an idea about necessary attributes for the design of the next prototype. We are also going to investigate the correlation between different muscles during the locomotion.

First we observe one of the muscles' rest length and actual length. Figure 5.10 shows that the transition phase from the stationary robot to locomotion is the first 8 seconds. This transition has a major difference from the locomotion which is a periodic motion. This muscle has to pulled to as low as 0.8 meters during the transition, but during the locomotion it varies between 0.9 meters and 1.05 meters.

After the transition phase, during the locomotion (8s - 20s in Figure 5.10) the rest length repeats the exact same pattern. This is because of the fact that the robot returns back to the same triangle surface after one revolution. Each revolution takes approxi-

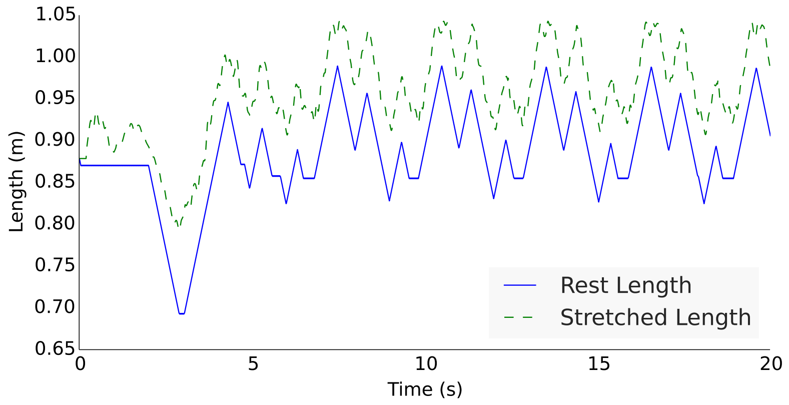


Figure 5.10: The rest length and the actual length of one of the muscles during the first 20 seconds.

mately 3 seconds and the policies that are selected are exactly same during this period. As expected, the actual length of the muscle differs from the rest length. The muscle stretches to apply tension. The stretched length of the muscle follows a similar pattern as the rest length, but there are minor differences between different cycles of the signal (i.e. at 10s vs 13s). This is due to the compliance of the structure with the ground.

Second, we investigate the lengths of all of the muscles. We interpret each muscle's length over time as a signal. We take 24 signals that belong to 24 muscles and illustrate each signal's mean and variance in Figure 5.11. The shortest length for the muscles is 0.65 m and the longest is 1.05 m. Each muscle has a different mean point but the variance for each signal is around 0.3m.

One can argue that these measurements are taken during locomotion in a specific direction and they might differ for another direction. This statement is partially true. While rolling in another direction the signals for each muscle will be different than the current signal. Indeed the signals will be the same, but they will be assigned to different muscles. This fact is based on both the symmetry of the structure and the design of the flop and roll algorithm.

Claim 1: Subsets of muscles will be using the same signal but with an offset.

In flop and roll algorithm, all the muscles have the same pool of policies that they select from according to the orientation. At each specific orientation, each muscle selects

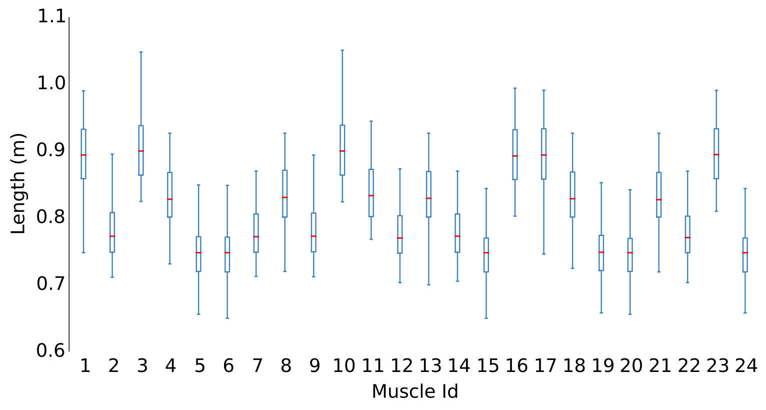


Figure 5.11: The actual lengths of the muscles during the locomotion.

a different policy from the pool. When the orientation changes, the robot lies on a different base and all the muscles change the policy that they use. Through this periodic locomotion, each muscle goes through a set of policies. We label the policies that the muscle i goes through as:

$$\pi_{i,t} \rightarrow \pi_{i,t+1} \rightarrow .. \rightarrow \pi i, t + z$$

where $\pi_{i,x}$ is one of the 48 policies that the pool contains. Let's assume that at orientation $o2$ another muscle (j) selects the same policy that the muscle i selected at orientation $o1$.

$$\pi_{i,o1} = \pi_{j,o2}$$

Since the flopping direction is known, the orientation of the structure will change the same way as it changed at orientation $o1$. Let's say that $o1'$ and $o2'$ are the orientations followed by $o1$ and $o2$ respectively. The policy for the muscle j at orientation $o2$ will be same as the policy for the muscle i at time $o1$

$$\pi_{i,o1'} = \pi_{j,o2'}$$

Using this step, we can prove that during the locomotion, if two muscles use the same policy at different orientations, they will select the same policies with the same order.

This shows that subsets of muscles will go through the same cycle of policies. As a result, their signals are going to be same.

Claim 2: Rolling to a different direction will cause the 24 muscles to use the same 24 signals in a different order.

Let's assume that the structure is rolling to a different direction. At a specific orientation, it has to make a flop to a different direction $d2$ instead of $d1$. The policy selected by the muscle i will be different than before. Let's label the policy selected by the muscle i rolling in direction d as:

$$\pi_{i,o,d}$$

. We know that:

$$\text{if } d1 \neq d2 \Longrightarrow \pi_{i,o,d1} \neq \pi_{i,o,d2}$$

.

On the other hand, at the orientation o rolling to $d2$ will cause another muscle j to select the same policy that muscle i would select rolling to $d1$. In other words:

$$\forall i \exists j, \pi_{i,o,d1} = \pi_{j,o,d2}$$

Moreover, using the claim 1 the next policies after the flop will also be same:

$$\pi_{i,o,d1'} = \pi_{j,o,d2'}$$

This shows that if the direction changes, for every muscle i there will be a muscle j, where the signal of the muscle i rolling in direction $d1$ will be same as the signal of the muscle j rolling in direction $d2$.

We investigate claims 1 and 2 by looking at the signals of the 24 muscles when they are normalized. Figure 5.12 shows 24 signals normalized on the top part. At first sight these signals look completely different. To discover the similarities between the signals, we use correlations between the signals. Figure 5.13 shows a 24x24 correlation matrix, where a dark color indicates complete match and a white color indicates complete mismatch. The matrix on the left is not easy to see the groupings between muscles. To better illustrate the data, we use hierarchical clustering between muscles according to this correlation matrix. The result of the clustering is shown in the middle. When the

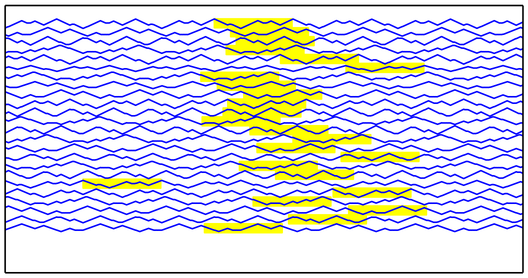


Figure 5.12: The 24 signals used by 24 muscles. Similar parts are highlighted according to their correlation.

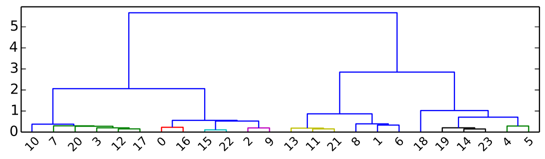
muscles are ordered according to the clustering, the matrix illustrates similar patterns between the muscles. The muscles are grouped by 6 to 4 groups. Within each group, the signals are similar to each other (i.e. the signals 0,2,9,15,16,22 are similar).

The best way to see this similarity is to order the signals according to these clusterings, then shift the signals according to their phase difference that results in the highest correlation value. Figure 5.14 shows the signals after clustering. As it can be seen there are 4 major clusters of 6. Within each cluster the signals are highly similar. This validates the fact that 6 muscles within a cluster use the same policies at the same order. This knowledge can be summarized as follows: An icosahedron tensegrity robot's muscles uses 4 different types of actuation to roll. All 24 muscles use phased versions of these 4 signals to have a smooth rolling locomotion.

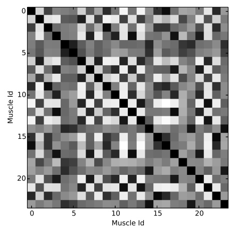
5.3.4 Feasibility of the locomotion

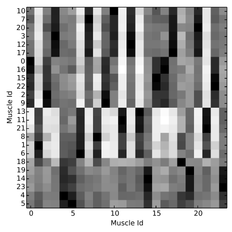
First, we test the learned policies with 3 stationary targets that consecutively becomes active one after the other. The goal of this experiment is to see the success of the algorithm in changing directions and also to see the robot's behavior during these sudden turns to different directions. The experimental setup and the trajectory followed by the robot can be seen in Figure 5.15. The robot rolls towards the first target for the first 20 seconds, towards the second target between 20s and 40s, and towards the third target during the last 20 seconds.

When we were analyzing the trajectories of the towards different directions, we ex-



(a)





(b) (c)

Figure 5.13: (a) the output of the hierarchical clustering, (b) The correlation matrix for the signals, (c) correlation matrix after y-axis is reordered with clustering.

plained the reason why the algorithm does not truly steer the locomotion, but instead prefers rolling in one direction then doing a sharp turn when necessary. The behavior observed here is exactly the same as before (Figure 5.7). When a sharp turn happens, the policies selected by specific muscles change providing rolling behavior to a different direction.

This transition between different direction causes higher or lower tensions for the muscles. This is an expected result for changing the momentum of a robot with a mass close to 10kg. For a rigid robot, this sudden change of momentum's effects could be concentrated on a specific joint. Since tensegrity structures are compliant, this impact is distributed to the whole structure. Figure 5.16 shows that the change in average tension is minor and the maximum tension value stays in a reasonable limit. The peaks are are 20 and 40 seconds when switching targets causes sharp turns. These numbers gives an idea about the applicability of the algorithm to the physical robot.

Next, we test the ability of the algorithm to overcome a terrain with different types of hills. After learning using maps with randomly-generated hills, we take the best policy and analyze its performance. The robot can continuously climb an inclined uniform terrain with a 20% grade (not pictured). In terms of nonuniform terrain with small

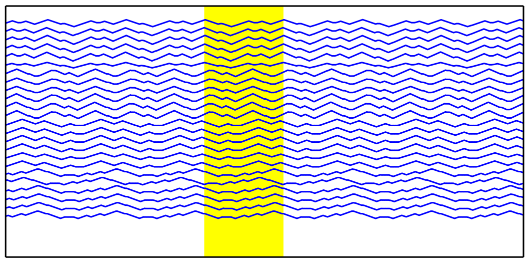


Figure 5.14: The 24 signals used by 24 muscles shifted and ordered according to the correlation.

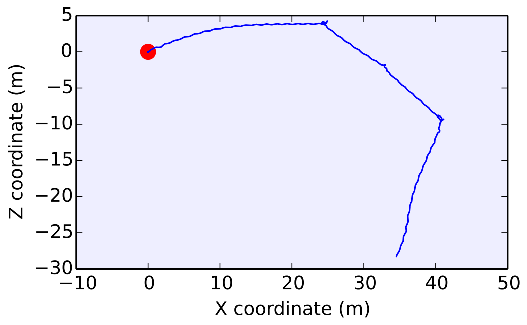


Figure 5.15: Trajectory followed by the robot with 3 consecutive targets located in different areas.

random hills, Figure 5.17 shows a 3D graph of the path of the robot with learned behavior. The robot climbs over small hills with grades of up to 33%. As a comparison, these hills have similar grades to the steepest streets of San Francisco.

Our next tests are about the applicability of the learned behavior with slower motor speeds. We train the robot using the default configuration of simple motors models with a constant speed of $0.2m/s$ independent of the load. We take the best policy and test it with motors as slow as $0.02m/s$. Figure 5.18 shows that the robot can still move in the desired direction. The learned behavior does not necessarily require fast motors. The slower motors mean slower rolling, but we obtained a distributed rolling algorithm independent of motor speeds.

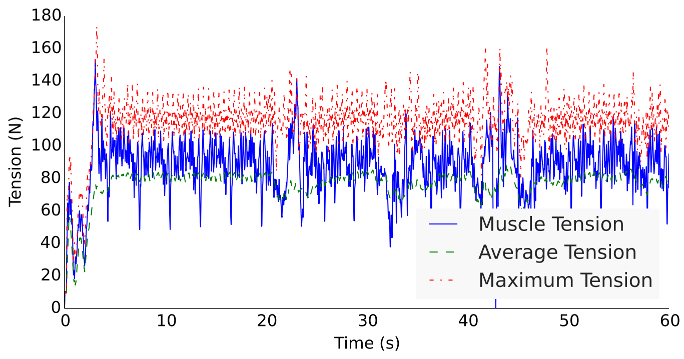


Figure 5.16: Tensions of the muscles while changing directions.

Another interesting behavior is the linear correlation between muscle speed and robot's rolling speed. If we consider a biped locomotion, running the motors with twice the speed will not necessarily result in a locomotion with twice the speed. Moreover, the problems with balancing the robot might cause locomotion to break. The reason that the rolling locomotion is linearly proportional with motor speeds is based on the flop and roll algorithm. The algorithm reacts according to the base triangle and deforms the robot to perform a flop. When the motors are twice the original speed, the time to reach the deformation to perform the flop takes is half the original time. The sequence of flops are the same but each one of them takes half the time. Since the robot's deformation is the same with faster or slower motor's the distance rolled with each flop is the same. As a result of these two facts, the robot performs twice the number of flops resulting in a locomotion that is twice the speed.

5.3.5 Robustness of the locomotion

The network of tensional elements that the tensegrity robot has provides natural compliance and robustness. For locomotion, we used coevolutionary algorithms to exploit distributed nature, and used historical average to discover more robust solutions. In this section, we are going to test the robot's reaction against external forces, broken muscles and actuator noise.

The advantage of a closed loop algorithm to an open loop one is to be able to

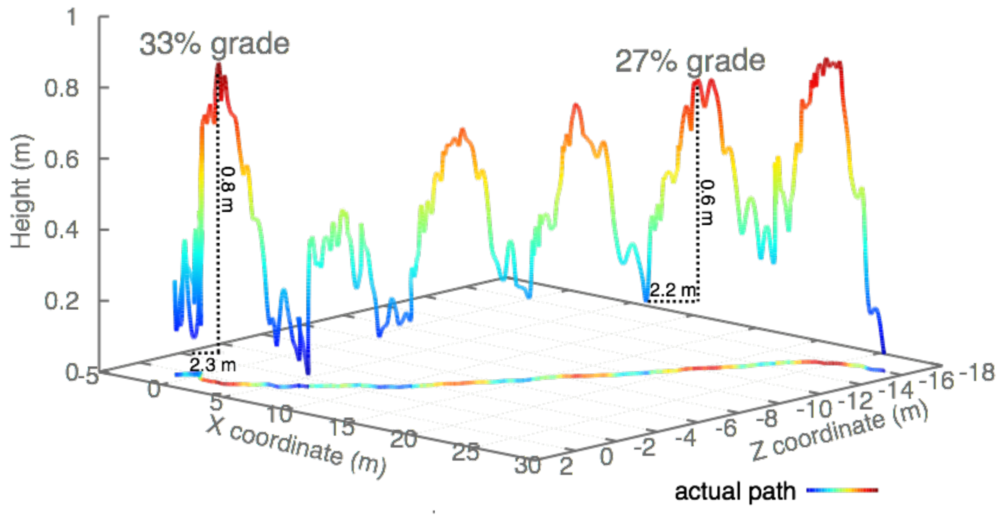


Figure 5.17: Flop and roll trained and tested on a terrain with steep hills. The robot climbs hills of up to a 33% grade.

handle different environmental conditions. In addition, tensegrity structures are also compliant and can handle external forces. Moreover, as opposed to legged locomotion, rolling locomotion does not have balancing problems. Here we test these 3 properties by applying extreme sideways impacts to the robot. Figure 5.19 shows the path of the robot when it is pushed sideways (twice) during the rolling motion. The robot smashed by the impulse rolls sideways but stabilizes and starts rolling towards its target again. This behavior shows two main behaviors: How the robot can handle external impacts and how the locomotion algorithm can keep working on rolling towards the desired target. These two conclusions are our major motivations behind working with a tensegrity robot and designing a closed loop locomotion algorithm.

While describing the tensegrity structures, the term robust was used many times. The nature of the structure has a network of tensional elements, where if one of them fails, the rest of the structure can compensate for the one. In our next experiment, we test if this same expectations work for a tensegrity robot and the locomotion algorithm. We disable one of the 24 muscles and observe the distance rolled towards the target per minute. Figure 5.20 shows that depending on the muscle that is disabled, the distance rolled by the robot varies between 30m and 80m. For some of the muscles the locomotion is not affected at all, for some the rolling speed is reduced. The robot is still operable

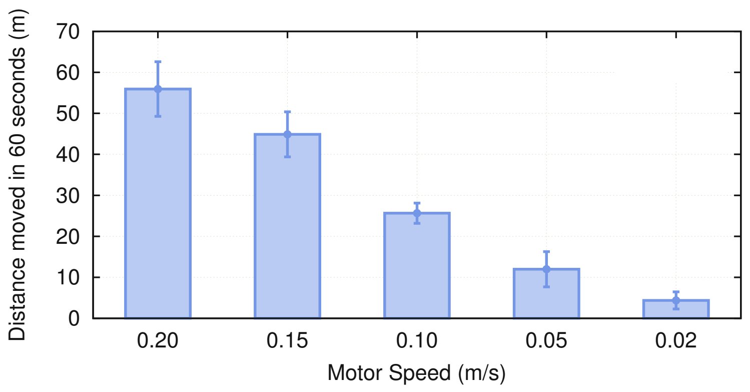


Figure 5.18: Flop and roll tested with slower motor speeds. Even if it is trained with a robot that has motors with 0.2 m/s speed, it can still roll with slower motors.

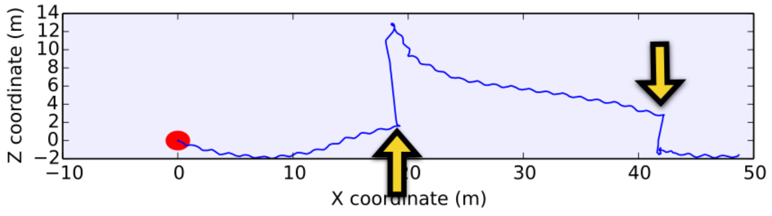


Figure 5.19: The 2D path followed by the robot with two intense unexpected external forces applied during the locomotion.

and performing locomotion in an underactuated setting.

The performance of the robot with disabled muscles also indicates that underactuated versions of icosahedron tensegrity robots is possible. Instead of 24 actuators, smaller number of active muscles combined with passive muscles can also lead to successful locomotion. As a result, reducing number of actuators can lead to lighter and simpler hardware designs. We leave further investigation of underactuated locomotion as a future work.

Through this dissertation, we work on developing and learning locomotion using simulation instead of a physical robot. In robotics community, one major argument against this approach is the noise that the real world bring to the actual robot. To respond, we investigate how the robot and locomotion behaves under different levels of actuation noise. The actuation noise defined in the simulator is the noise applied at the

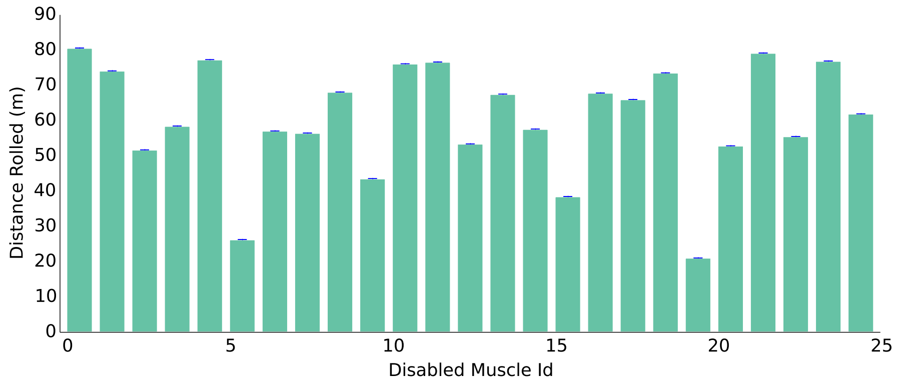


Figure 5.20: The effect of disabling one of the muscles on rolling locomotion.

muscles while pulling or releasing the cable. If the noise level is $x\%$ the change of length for the muscles is $z \pm z.x/100$ where z is the motor speed (0.2 m/s for our specifications).

Figure 5.21 shows how trajectory of the robot is effected when the actuation noise is increased. With small percentage of noise, the trajectories diverge but the overall rolling direction is the same. With increased amount of noise, the resulting trajectories are similar to serpentine figure. Compared to the rolled distance, disturbance is minor. We can claim that the robot can easily take up to 100% noise without problems, but minor steering occurs. The rolled distance and the overall direction is same. After 200% noise, unintentional steering becomes more obvious and sharp turns become necessary after high deviation. Considering the hardware setup of SUPERBall, an actuation of level 100% is highly unlikely for the physical robot. As a result, actuation noise causes minor deviation, but is not a major problem for rolling locomotion.

Designing the flop and roll, we did not consider a steering option for the locomotion. Instead, we explained that the robot does sharp turns when needed. The serpentine figure caused by the noise shows that it is possible to have a rolling locomotion that steers and steering is definitely related to the contracting speed of the muscles. One possible approach to provide steering is to slow part of the muscles. While investigating the locomotion, we discovered that there are 4 types of muscles during the locomotion, and these muscles are grouped on different sides of the robot. We believe that slowing down a particular group of muscles (such as the ones located on the right), can result

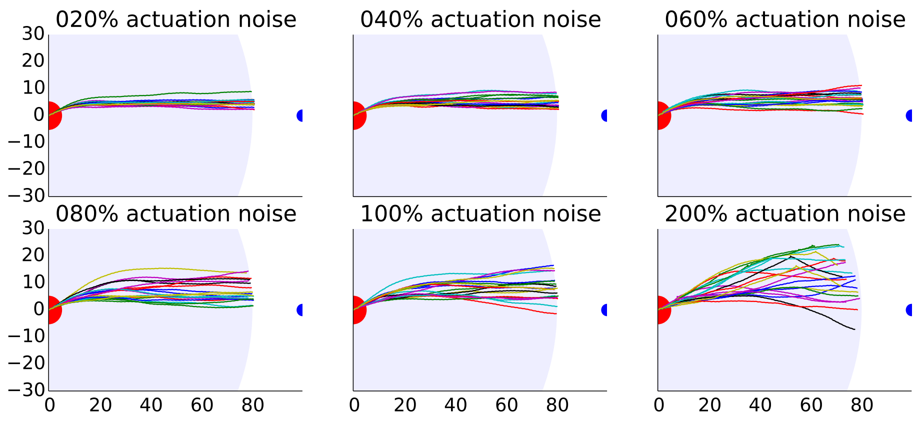


Figure 5.21: The effect of actuation noise on the trajectory followed by the robot.

in steering behavior. The methods to add steering to rolling locomotion is definitely an interesting future work.

5.4 Conclusions

In previous chapter, we showed an open loop rolling algorithm for tensegrity robots. We showed that rolling is feasible, and presented a general roadmap to develop rolling behavior for tensegrity robots. On the other hand, the resulting behavior was not controllable and it did not have any sensory information to handle different environmental conditions. In this chapter, we presented flop and roll, a rolling algorithm that uses contact sensors as sensory input.

While designing flop and roll, to take advantage of the symmetric nature of tensegrity structure, we defined policy pooling, a method to simplify the rolling problem into simple flops. The agents that are defined as muscles of the structure, used the behaviors learned for a single flop interchangeable depending on the orientation of the structure. These simple flops are optimized according to the resulting rolling behavior. As a result, the problem learned by coevolutionary algorithms are much simpler than the signals that are learned for open loop controls in previous chapter. The sensor information required by flop and roll is minimal. The robot senses the desired direction to roll and which side the robot lies on using contact sensors that are at the end of the robot. Using this information, the agents decides which policies they are going to use from the pool.

The problem that we address with flop and roll is more complex than open loop controls. The robot rolls in a desired direction and can handle unexpected terrain conditions. Despite these challenges, we showed that flop and roll learns faster and a better rolling behavior than open loop controls. We first analyzed the learned behavior by looking at the resulting gait and muscle movements while rolling to different directions. Our experiments show that the robot follows a symmetric gait while its muscles' behavior can be classified into 4 different sets. We observed the similarity of the resulting tensegrity locomotion with bipedal locomotion by looking at the center of mass and footprints during the rolling behavior.

As another set of experiments, we discussed the robustness of the behavior. The robot could handle different terrain conditions and unexpected external forces without any problems. We also introduced artificial actuation noise and observed that the robot could handle reasonable noise levels without any problems.

Flop and roll is the first directional rolling locomotion algorithm for tensegrity robots. It is based on coevolutionary algorithms, historical average fitness shaping and policy pooling. We used NTRT and the model of SUPERBall, a physical icosahedron tensegrity robot that is under development. The definite future work for flop and roll is its application to the physical robot, when it is built. For this procedure, feasibility, simplicity and robustness of the algorithm are critical. The experimental results that we presented here give a good insight on the way for rolling locomotion for a physical tensegrity robot.

Chapter 6: Multi-robot coordination using Reinforcement Learning and Reward Shaping

Chapters 4 and 5 used multiagent learning to learn locomotion for a tensegrity robot. Different agents controlled different parts of the robot to establish locomotion. During this process we used coevolutionary algorithms and fitness shaping to solve credit assignment problem. In this chapter, we switch to a higher level coordination problem in the field of tensegrity robots: Learning multi-robot coordination.

In a multi-robot coordination problem, the robots navigate to observe points of interests (POIs) placed in an environment. The problem is episodic, meaning that the robots have a constant amount of time after they are placed in the environment. There is a team of robots and multiple POIs to observe. The goal is to maximize the amount of total observation in an episode. At each time step, the observation is calculated as:

$$G(z) = \sum_{p \in POIs} \max_{a \in Agents} (0, \beta - min(distance(a, p)) \tag{6.1}$$

where p represents POI in the system, a represents the agent which minimizes the distance, and β represents the maximum distance that a POI can be observed from.

The inspiration behind multi-robot domains is the surveillance problems. Imagine a team of robots patrolling an area where some parts of the map is marked as critical. These robots can be rovers on a space mission, or UAVs on a border patrol. In such a situation, the communication between the robots may not be reliable, or a centralized control might not be possible. Each robot has to be independent, but they have to learn to cooperate to maximize observation made at the environment.

To solve such a problem, we use multiagent reinforcement learning which was explained in Chapter 2. Although the team of robots are maximizing the amount of total observation, using this as a reward causes the credit assignment problem. Instead, using shaped reward for each agent according to its contribution can improve the learning performance. Difference rewards addresses this problem, but it also brings other challenges such as applicability and information requirements.

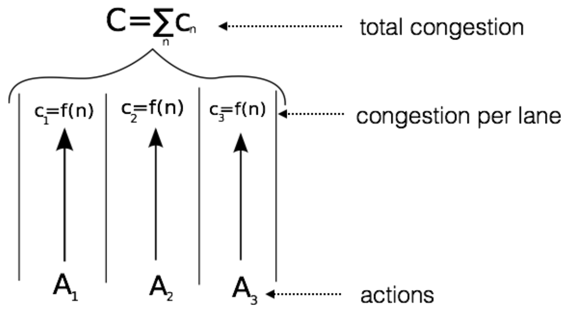


Figure 6.1: Highway example as a congestion problem. Total congestion is calcuated as the sum of the congestion in each lane.

Through this chapter, while working on a multi-robot coordination problem, we are going to address two different aspects of shaped rewards: Information requirements and handling task allocation. Section 6.1 introduces interaction space of shaped rewards to decrease information requirements. Section 6.2 develops a dynamic task allocation method using difference rewards for multitask problems.

6.1 Interaction Space for Shaped Rewards

The key concept of reward interaction space is to determine the necessary amount of information required to effectively compute a particular agent reward. The RIS of an agent in the environment can be explained as a part of the environment where possible explicit or implicit interaction with another agent can change agent's utility for the system. As an example, if we consider a typical congestion problem such as driving on the highway, each agent choose a lane to drive. The system congestion depends on the number of agents in each lane and is calculated as the sum of the congestion in each lane. Figure 6.1 illustrates the the problem with three lanes.

Now, let us carefully analyze this example. Clearly, in a traffic problem, one can argue that congestion in two lanes is interdependent, because there can be a flow from one lane into another. However, at any given time, if congestion is computed as a sum of the congestion in each lane, the lanes independently contribute to the full system reward. Although movement between lanes negatively impacts congestion, it does so only because it directly increases congestion in both lanes if traffic has to slow to accommodate that lane change. Considering the given example, a local reward reward function would only require information about the current lane while a global reward function would require information about the complete environment. The difference rewards are defined in terms of the global reward function which requires full information. However, it is not apparent how much of that information is actually necessary or used by difference rewards. To goal of the interaction space is to find this unknown.

6.1.1 Interaction Space

To be able to formalize the concept described above, we will use mapping techniques from probabilistic robotics. A map of the environment is a list of objects with their properties represented. It can be feature based or location based [79]. To define RIS, we will use location based map of the environment defined as:

$$m = m_1, m_2, ..., m_N \tag{6.2}$$

where $m_k = m_{\{Loc_k, Inf_k\}}$ represents information I about a specific part of the environment (location Loc).

As an example, simplest location based maps are occupancy grid maps. Assuming that the environment is a 2D world,

$$m_k = m_{\{x,y,o\}}, o \in \{0, 1\} \tag{6.3}$$

where $o = 0$ means location x, y is free, and $o = 1$ means location x, y is occupied. Another example can be a height map, where z means the height of the environment at location x, y.

The attached information represented by z can be any type of information such as obstacles, other agents, or any other objects. As an example, if we want to construct location based map of the highway example, one possible representation would be using lane 0, 1 or 2 as locations, and using number of agents on that lane as the attached information.

Since we have the environment represented as a list of location-property pairs, we can define RIS using the same idea. RIS of an agent i existing at a location l is a set of locations where the attached information affects the reward of the agent i. Formally,

$$RIS_{\{l\}} = \{loc \in L | R_i(M) \neq R_i(M_{-loc})\} \tag{6.4}$$

$$M_{-loc} = \{m_{\{l,inf\}} | l \neq loc\} \tag{6.5}$$

where l is the location of the agent, L is the set of all possible locations, R_i is the reward function provided to the agent i, M is the complete information about the environment and M_{-loc} is all the information about the environment except location loc.

Considering the highway example and reward structures explained before (global and local), the value of the reward function with and without an agent would always change while using global reward. So every possible location, therefore every other agent is within RIS of an agent using global reward. On the other hand, for the local reward function, two scenarios with and without any other location (lane) returns same local reward for the agent because local reward is only related to the congestion at the lane the agent is in. So, the RIS of the agents using local reward is limited to the lane that they are in at a specific time step. This means that, if an agent is using global reward its RIS is the whole environment, if it is using the local reward its RIS is the lane that the agent is in.

Although it is easy to find RIS for these two reward functions, it is not straightforward to find it for different shaped reward functions. Considering the difference rewards explained before, the formula is $D_i = G_z - G_{z-i}$. To be able to decide if another agent is in the RIS of an agent i, we need to calculate D_i once considering the existence of agent j and once when the information about that lane is missing. For a scenario where the agents i and j are in different lanes,

$$D_i = G_z - G_{z_{-i}} = \sum_{l \in lanes} c(l) - \sum_{l \in lanes} c_{-i}(l) \tag{6.6}$$

where c is the congestion reward function for a given lane. Since, the information about other lanes does not change with or without agent i, $c(l) = c_{-i}(l)$ when l is not equal to the lane that agent i is in. So equation 6.6 reduces to:

$$D_i = c(x) - c_{-i}(x) \tag{6.7}$$

where x is the lane that agent i is in.

If we follow the same exact procedure for $D_{i_{\{-j\}}}$ it can be shown that the same reduction results in $c(x) - c_{-i}(x)$. As a conclusion, the equality $D_{i_{\{-j\}}} = D_i$ holds when j and i are at different lanes. This means that agents in different lanes are not in the RIS of each other with respect to difference rewards. Using similar derivation, one can easily derive that this equation does not hold when two agents are at the same lane, which concludes that RIS of an agent is limited to the lane that it is in. This example shows that for agents using the difference rewards method, the RIS is only the lane that the agent is in, not the whole environment.

6.1.2 Discovery of the Reward Interaction Space

In the highway example explained above, the problem is simple enough to discover the Reward Interaction Space manually. We have been able to prove that RIS of an agent that uses difference rewards is its current lane according. On the other hand, in many domains it is difficult or not intuitive to define the limits of the RIS. To be able to use it in different domains, presenting a method for automated discovery of the RIS is as important as the concept. Given a domain, the RIS can be found by preprocessing the environment and the reward function before the learning process.

The goal is to try to find RIS of an agent when it is on a specific location. For the given problem, the information attached to a location is existence of another agent. We create a possible snapshot of the environment with two agents, compare it to the case after removing the second agent. The difference shows if existence of the second agent affects the reward of the first agent (Figure 6.2). If it affects, the location of the second agent is included in RIS for the location of the first agent. Executing the same comparison for every possible location in the map gives us the RIS of an agent for that location.

Formalization of the procedure is straightforward. First, let's define the effect of agent j on agent i. It can be defined by the subtraction of the reward of the agent i with existence of the other agent (j) and without the other agent (j). If we formulate this description, we get:

$$E_{i \leftarrow j} = R_i(z) - R_i(z_{-j}) \tag{6.8}$$

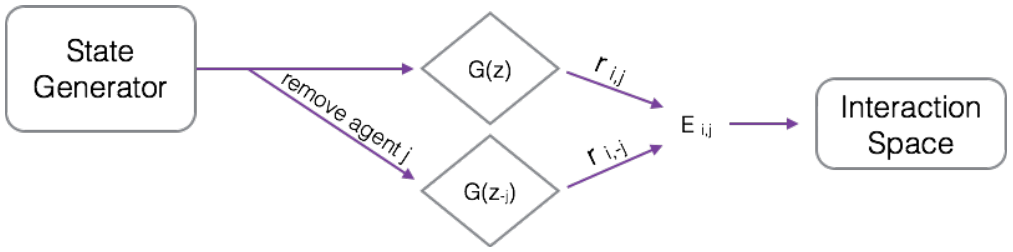


Figure 6.2: Sketch of the method to discover Reward Interaction Space of an agent with respect to a reward function. We take a snapshot of the environment (z), feed the reward function with z and z_{-j} (without agent j) and subtract the outputs to calculate the effect.

Algorithm 4: Algorithm for automated discovery of Reward Interaction Space for congestion domains

```
Input: Locations,Reward Function
Output: Reward Interaction Space for each location
foreach $loc1 \in Locations$ do
    $IS(loc1) = \emptyset$;
    foreach $loc2 \in Locations$ do
        Place $agentA$ to $loc1$ and $agentB$ to $loc2$;
        if $E(AgentA, AgentB) \neq 0$ then
            $IS(loc1) \leftarrow loc2$
        end
    end
end
```

where $R_i(z_{-j})$ is the reward of agent i at environment state z without including agent j.

The equation 6.8 is similar to the last part of the equation 6.4, but instead of locations, we use agents. The function $E_{i \leftarrow j}$ defines the effect of agent j on agent i. To be able to find the whole RIS for the agent i (i.e. when another agent affects the agent's rewards), the algorithm needs to check the values of E for different placements in the environment. For each possible combination of locations of agent i and agent j, if the effect is not zero, the second agent's location is added to RIS of the first agent. The general flow of this procedure is shown in algorithm 4.

Let's go through the simple highway example and use local reward (same lane) as the reward function. We can compute the RIS by putting one car to one of the lanes and measure the effect of the other cars in other lanes using the formula presented above. If we put one car to the left lane and another one to the right lane, if the agents use local reward, the effect of the second car on the first car will be zero. On the other hand, if we put two cars to the same lane, this formula will give a positive number, meaning that two cars that are at the same lane are in each other's Reward Interaction Space. As a consequence, this algorithm "discovers" that an agent only needs information about the agents in the same lane as itself, not about the other lanes. As it was explained in Section 6.1.1 the same conclusion also holds for difference rewards.

At the end of this preprocessing, we get the whole RIS for the environment and reward function. As a conclusion, depending on the agent's location, we know what type of information the given reward function requires to achieve full performance.

6.1.3 Complexity and Constraints

The method explained uses preprocessing power to decrease information requirements of the agents for a learning problem. As the time spent for learning is important, we have to make sure that the algorithm does not use excessive processing.For Algorithm 4, one can see that the complexity is $|L|^2$, where L is set of locations in an environment. Given a learning problem and MDP, location is not a defined concept. As explained in Section 6.1.1, the algorithm needs a location based map of the environment. Although it is called location based map, it does not have to be an actual map or actual locations in the environment.

In this context, the concept of map needs further explanation. A map of the environment is the set of information that defines the environment. If it is possible to decompose the information about an environment into independent subsets, and if each agent can only be part of one subset, this decomposition of the information can be called a map. Similarly the subset that contains agent's information is called the location of that agent. As an example, in the highway problem, the information is decomposed into 3 different subsets called lanes and each agent belongs to only one of the subsets.

If the domain is a congestion domain where the agents take actions without states and the global reward is calculated according to their last action, the locations can be defined as actions. In that case, the complexity of the given algorithm is $\mathcal{O}(A^2)$ where A defines the number of possible actions to take. Note that the calculations do not depend on the number of agents, which is an important feature for a multiagent problem.

Although the examples given were action-reward type congestion domains (without states), the same algorithm can also be used in domains with state action pairs. Locations are not the actions, but they can be defined in a different way. For example, in a gridworld, locations are naturally defined as the grid cells that the agents are in. On the other hand, in a 3D simulation problem, one can discretize the environment and use the same algorithm by using each tile as a location. In both types of domains, the complexity will be $|L|^2$, which is significantly low compared to learning time of a multiagent learning problem. Moreover, it is always possible to decrease the resolution of the mapping to lower the preprocessing time. Number of locations can decrease by associating more information with each location. On the other hand, by decreasing the resolution, the information provided to the agent's reward function might contain more information than it needs. This trade off can be considered as changing size of the filter while classifying the information required for the reward function.

6.1.4 Experimental Results

While presenting results we will use 3 different problems with different complexities. At first, we are going to show the results of the interaction space in the simplest case: a bar problem that the agents learn to attend to a bar on different days. Next, we will step into the environment with multiple robots and multiple POIs as described, but we will isolate the highest level of the problem from internal dynamics. Each robot will choose a spot in the map as desired location. The observations and the rewards will be calculated according to these decisions.

The experiments in this chapter contain multiple tensegrity robots and other objects in the environment. Since we are using multiagent learning, the methods require thousand of simulations to learn. To overcome this issue, we use a lower fidelity version of the simulator. The robots roll around with a fixed speed, but lower level interactions between the robot and the environment are ignored. This abstraction is coherent with the locomotion results that we found in Chapters 4 and 5.

6.1.4.1 Bar Problem

The bar problem is a commonly used multiagent congestion domain based on attendance at a bar on different days of the week [5]. In this modified version, at the start of every week, all the agents in the system choose the day that they want to attend to the bar. After the end of the week, according to the distribution of the agents, and the capacity of the bar, the reward of the week is calculated using:

$$\sum_{d \in Days} f(a_d) \tag{6.9}$$

where a_d represents the attendance of the day d and $f(x) = x\,e^{-(\frac{x}{c})}$ represents the function for the reward, with c representing capacity of the bar. Many different functions can be used for f, but the preferred setup is to have a maximum value for the capacity of the bar, because when the bar is over capacity it is crowded and less enjoyable and when the bar is empty it is not enjoyable either.

The described bar problem is similar to the highway example presented above. Considering the similarities of the problem with highway example, it can be expected that RIS of the agents using difference reward will be limited to the day that they attend the bar. As expected, the results of the algorithm agree with this fact showing that effective area of an agent is the day that the agent attends the bar. According to this result, if the information that difference reward function gets is limited to the same day as the agents attend, the agents will perform at the same level, but the information usage will be decreased to only one day of the week.

Comparing the performance of the agents, previous work has already shown that Difference Rewards makes the agents perform better than System Reward. The purpose of using RIS is to use only partial information and match the level of performance that difference rewards give. Looking at the Figure 6.3, it can be seen that agents with partial information (observing only the day that they attend), can perform as well as full information (observing the bar all week). Using the RIS, agents can perform at the same level while observing 1 day instead of 7.

	M	T	W	Th	F	S	Su
M	+	-	-	-	-	-	-
T	-	+	-	-	-	-	-
W	-	-	+	-	-	-	-
Th	-	-	-	+	-	-	-
F	-	-	-	-	+	-	-
S	-	-	-	-	-	+	-
Su	-	-	-	-	-	-	+

Table 6.1: Result of automated discovery of the RIS for the bar problem. As expected, the RIS of each day contains only the agents who attended the same day.

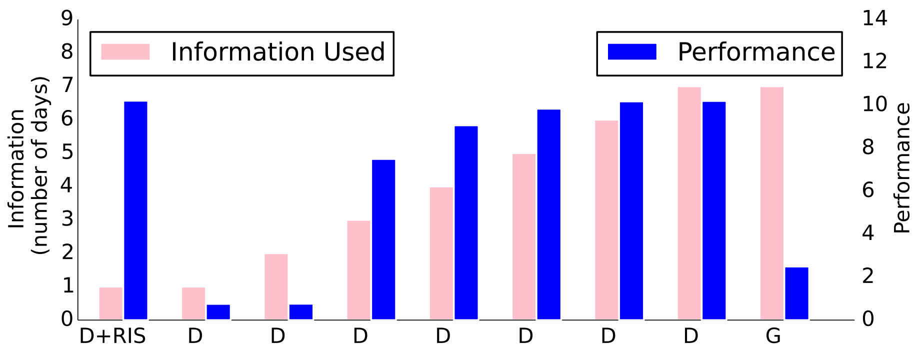


Figure 6.3: Performance of the agents in the bar problem with different levels of information. Agents with difference rewards calculated only with the Reward Interaction Space (observing the same day) can perform as well as the agents using difference rewards in fully observable domain.

6.1.4.2 Abstract Multi-Robot Problem

Multi-robot problem is a domain containing points of interest (POI) that agents need to observe by navigating through the environment. The agents try to learn a team behavior to increase the number of observations made by the team at every time step. In this section, we use abstract actions to make the problem simpler. Unlike typical robot domains, instead of choosing which direction to move, the agents directly choose a position on the environment signifying where they want to move. After all the agents decide their actions, we assume that the locomotion is handled at a lower level. At the next step, the agents are rewarded according to their action selection, the problem does

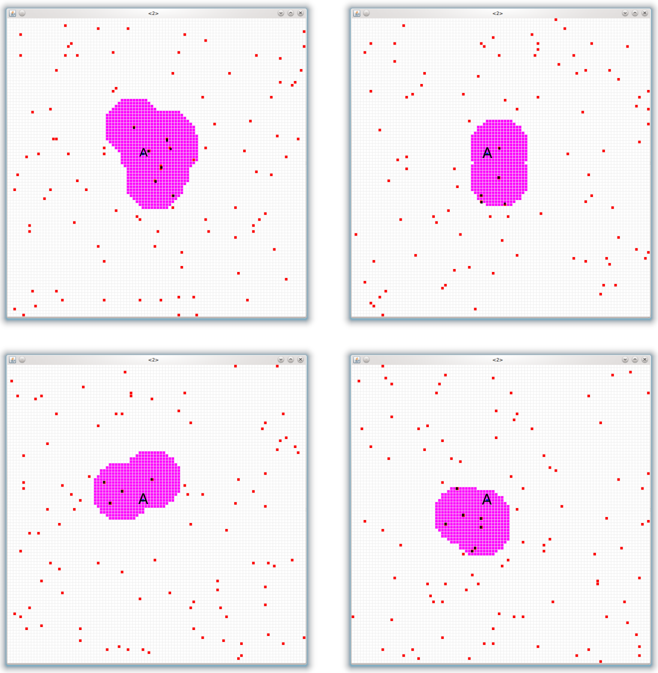


Figure 6.4: Illustrations of the RIS for 4 different POI distributions. The agent is at the center of the environment, the POIs are distributed randomly. The black dots are the POIs in the Reward Interaction Space of the agent, red dots are the POIs outside the Reward Interaction Space of the agent. Purple Area is the RIS of the agent.

not contain any states. This abstraction simplifies the problem to a one-shot domain, but the concept of observation according to the distance makes it more complex than the bar problem.

The main goal for the agents is to choose the best spots to maximize observation of POIs. Observation of the POIs is defined according to the distance. Each POI is observed by the closest agent, and the observation value of that POI is determined according to the distance between the POI and the closest agent. For the whole system, the global reward that the agents try to increase is the sum of the observations for all of the POIs, defined as:

The difference reward discussed above is defined as the subtraction of the system reward with the agent and system reward without the agent . As shown by previous work, difference rewards result in better performance than global or local rewards, but to be able to do the calculation, each agent needs complete information about the positions of all the agents.

This version of the rover domain contains the challenges of a cooperative domain (in fact it mostly removes the navigation-related difficulties, not the coordination-related ones). Compared to a simple congestion problem, the number of possible actions that the agents can take is significantly higher (height * width of the environment). Moreover, the environment has a two-dimensional structure containing a distance metric between choices. Since positioning can affect neighbor states, possible positions of the agents are not independent (as opposed to the bar problem). As a result, the RIS is not as straightforward to compute and is composed of an unknown area surrounding the agent.

If we try to calculate the RIS manually, we have to analyze difference rewards and the system reward in detail. $G(z)$ was defined as a sum of observations of POIs. Assuming that the function for measuring the observation for a POI p by an agent a is represented by $f(p, a)$, combining f, Equation 6.13 and Equation 6.1 gives:

$$D_i = \sum_{p \in POIs} \max_{a \in Agents} f(p, a) - \sum_{p \in POIs} \max_{a \in Agents_{-i}} f(p, a) \tag{6.10}$$

The terms of first element and the second element only differ at the places where agent i is the closest agent to the POI p (Case A), so it is 0 in all the other cases.

$$D_i = \sum_{p \in POIs} \left\{ \begin{array}{c|l} \max\limits_{a \in Agents} f(p, a) - \max\limits_{a \in Agents_{-i}} f(p, a) & \text{Case A} \\ 0 & \text{else} \end{array} \right. \tag{6.11}$$

For most of the POIs, agent i is out of the range, which gives the ability to cancel elements from both of the terms. Canceling every POI that agent i cannot affect, reduces Equation 6.10 to:

$$D_i = \sum_{p \in range(i)} \max_{a \in Agents} f(p, a) - \max_{a \in Agents_{-i}} f(p, a) \tag{6.12}$$

This reduced definition of the difference reward does not require any information about the far POIs and about the agents that are not in the range of this small set of POIs. As a description, the resulting information needed is composed of the agents that are in range of the POIs for which the agent i is also in range.

On the other hand, using the algorithm provided,we can discover the RIS for an agent

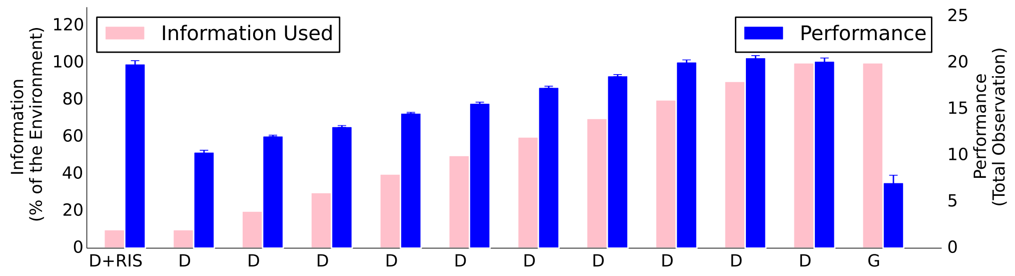


Figure 6.5: Performance of the agents in abstract multi-robot problem with different levels of information. Agents using difference rewards and RIS can perform as well as agents using difference rewards in fully observable domain while using 90% less information.

for each possible scenario. As explained in Section 6.1.3, this process has only $\mathcal{O}(A^2)$ complexity. Moreover, the method gives exact shape of the RIS that can vary depending on the positions of the agents and POIs. Figure 6.4 shows 4 different cases where the algorithm finds RIS with different shapes. After processing every possible snapshot, the algorithm results with different shapes of RISs, but they are all within an area of radius 20.

As the RIS is discovered, the next set of experiments measure the performance of the agents with the limited information. Figure 6.5 shows that the agents can reach same performance level only using the information about their RIS. In the setting used in these experiments, it allows the agents to get the same performance with 10% of the complete information, but for bigger environments this ratio will be even less, because RIS of the agent for this domain does not depend on the size of the environment.

We compare three different rewards (Difference rewards , D+RIS, Global) at different levels of available information. We compare the performance that the agents reach after converging at each scenario and reward combination. We expect that the agents that receive information about their RIS and use D should perform the same as when they use D and receive full information about the environment. As expected G does not provide much, but leave it to show the importance of reward shaping in such a system. As we can see in Figure 6.5, D+RIS uses only 10 % information, while doing 20 units of observation. This result holds for D when there is 100% information. As expected, agents using global reward are way below compared to these two options.

To show the significance of the performance that D+RIS reach with small amount of information, we test D with less amount of randomly selected information. As it can be seen, lack of information hurts the performance of the agents with D. Interestingly, the agents using D have slightly better performance around 80% information, which is caused by a peculiarity of this domain: in this case, ignoring some agents forces agents to prefer positions where they perform a backup function for the "unseen" agents. This in turn causes a slightly better spread of the agents on the POIs (note this behavior is also observed in Figure 6.6). When the amount of information drops to 10% (size of RIS), the advantage of using a shaped reward is completely lost. This experiment shows that if the information provided to a reward function covers the RIS, we can take full advantage of the reward shaping function while reducing the information used. This reduction is 90% of the environment for this specific problem.

6.1.4.3 Stochastic Multi-Robot Problem

In previous domains, all the agents pick an action (choose a lane, a day or a location), and the agents receive a reward according to the distribution of these actions. This is not always possible in real world problems. As an example, considering a more realistic version of the highway problem, if an agent is in the leftmost lane and chooses to drive in the rightmost lane, it changes the lanes one by one instead of jumping to the rightmost one. During this transition, it interacts with all the intermediary lanes as well as its original and final lane. To be able to test this kind of domains, we introduce a stateful version of the rover domain.

We now introduce a stochastic problem where robots take primitive movements at the environment instead of being teleported. Though seemingly simple, this difference in the domain causes a significant change, because the agents take actions to change their locations and they are judged according to their new position (defined by previous location and action), not according to only their action. According to the definition, the problem becomes a stochastic game where state and action returns new state and the reward.

In this case, we use an abstract decision layer that consists of multi-step actions. The decision mechanism is converted to a two layered hierarchical one, where the top layer (responsible for cooperation) decides to the desired location and the second layer

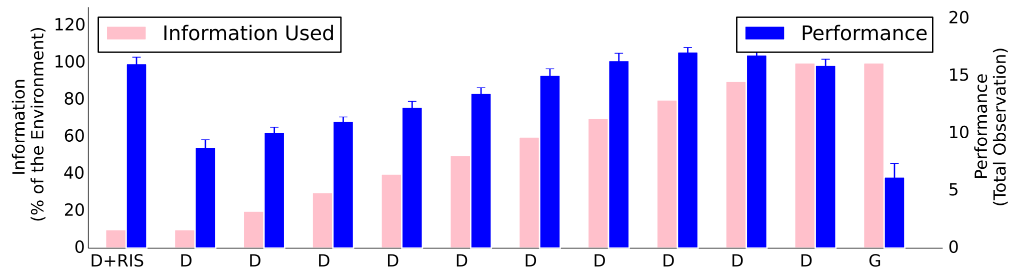


Figure 6.6: Performance of the agents in stochastic problem with different levels of information. Agents using difference rewards RIS can perform as well as agents using difference rewards in fully observable domain while using 90% less information.

(responsible for navigation) is responsible for moving towards that direction. Although it is possible to use learning approach for both layers, we assume that the agents have a policy for the navigation system. This assumption is actually supported with the results that we presented on tensegrity locomotion (Chapters 4 and 5).

Although the problem looks similar to the abstract version, there are significant differences. The agents get the reward according to their location, and navigating to the chosen positions take different timesteps for each agent. Considering that learning layer decides preferred location and it takes time to go to the destination, the agents will not be able to get the appropriate feedback at every timestep. We used a fixed timestep interval between two decisions and rewards. This period does not guarantee that the agents will end up at their preferred locaton, but it gives enough time to perform significant displacement.

One more time, we compare same three different rewards (D, D+RIS, Global) at different levels of available information. The problem is stochastic and more complex than the previous experiment. As expected, Figure 6.6 shows that the agents that receive information only about their RIS and use D performs the same as the agents that use D and receive full information about the environment. As expected G does not provide much, but leave it to show the importance of reward shaping in such a system. As we can see in Figure 6.6, D+RIS uses only 10 % information, while doing 16 units of observation. This result holds for D when there is 100% information. As expected, agents using global reward are way below compared to these two options. To show the significance of the performance that D+RIS reach with small amount of information, we test D with less amount of randomly selected information. The lack of information hurts the performance of the agents with D. Same as the previous experiment, the success of D+RIS can be seen by looking at the difference at the 10% information level: the agents still have full advantage of using the shaped reward while using only small amount of information.

6.1.5 Conclusions

In this section, we addressed the information requirement problem of reward shaping functions. Despite their definition, the reward shaping methods can actually require less information than the full environment. We defined the set of information that is actually needed as reward interaction space and derived a method to discover it for a given problem and reward shaping method. We used the multi-robot scenario to show that with 10% of the environment, the agents can still benefit from reward shaping methods to tackle the credit assignment problem.

6.2 Dynamic task allocation with Difference Rewards

In chapters 4 and 5, we handled locomotion for a single tensegrity robot using multiagent learning. In this chapter, we moved into coordination of these robots when there are multiple of them working on a common goal. We address two problems of multi-robot coordination and multiagent learning. Last section addressed the problem of information requirements for reward shaping for a team of robots learning to observe an environment with points of interests. In this section, we address the problem of handling task allocation using the same setup. The robots are learning to coordinate and there are multiple tasks that have to be handled by different agents. To handle these types of problems we will introduce a dynamic task allocation method "HELM", that is based on multiagent reinforcement learning and difference rewards reward shaping.

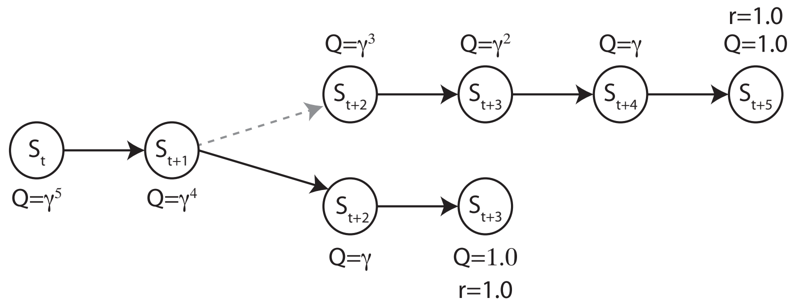
Problems containing multiple tasks can quickly become complex for a learning agent. Considering that each task is part of the state representation of the agent, the resulting state space grows exponentially in terms of number of tasks. Moreover, taking an action and getting a reward can be confusing due to not knowing the source of credit in terms of different tasks. Looking from a human perspective, for such problems with multiple tasks, giving a decision on a high level task and then choosing a primitive action is more intuitive for a human being.

For problems with multiple tasks to choose from, the most intuitive way of structuring an agent would be using a higher level decision mechanism for the selection of the task and a lower level decision mechanism that decides on the primitive action. Although this schema would help reduce the complexity of the problem, it introduces two levels of concurrent learning. When two levels of concurrent learning are involved, learning can become hard with one reward corresponding to the consequences of two actions taken on different levels.

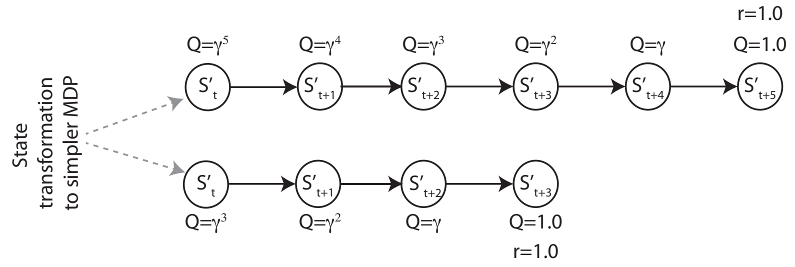
In a hierarchical schema, if we summarize each level's responsibility, the lower level is responsible for how good each action is for a specific task, and the upper level is responsible for how good selecting each task is in a given situation. The problems can be heterogeneous or homogeneous in terms of tasks. For this method, we consider homogeneous case where the agent has a selection of similar tasks that has same MDP with different parameters. For example it can be picking up an object, but the place of the object can be different in different tasks. The tasks are the same, but the parameters or the conditions are different. In such type of problems, all these choices can be handled using one learning: picking up an object from a given point. At this point, we decompose the problem into a simpler MDPs. As reinforcement learning is used for learning this MDP, the lower level learning mechanism contains a perfect feature that can be used for higher level evaluation: Q values. The initial research for this work was published in [33]

6.2.1 HELM

The reinforcement learner has a Q table which contains Q values of each state action pair, and these values are mainly used for decision making between different possible actions in a specific state. These values not only signify the best action for that state, they also propagate to the other states which contain a transition to that state. Each of these Q values are actually estimations of the expected sum of the discounted rewards starting from that state and following the current policy. These estimated values are not only beneficial for action selection, they give the agent perfect opportunity to judge how "good" a specific state is. This property results in a Q table where values become



(a) State action flow for traditional RL



(b) State action flow for two level learning schema

Figure 6.7: Comparison of flat learning and task decomposition approaches for a problem with two goal states that are 3 and 5 steps away. Since the flat learner propagates the values from the best next state, the current value of the state is γ^3. The two level learner has two different tasks where corresponding Q values of the decomposed versions of the current state are γ^5 and γ^3 with respect to different subtasks. The subtask that is closer has a higher Q value.

higher when the agent is closer to a goal.

If the agent uses flat learning, for the problems with multiple subtasks, Q values are propagated from the closest goal state because the algorithm propagates the value of the best state. The Figure 6.7-a illustrates this propagation at the decision point. On the other hand, in a hierarchical schema, considering that each subtask is handled independently, Q values will be propagated separately instead of merging (Figure 6.7-b). This leads to independent Q values for different tasks. Looking at the overall picture, the state given to the agent is decomposed into a different state for each different task. Each of these states, are part of a simpler MDP which only contains one task. As a conclusion, the learning agent is able to check the Q values of these decomposed states. When the learning for a task is converged to the optimal policy, these Q values for each task will increase depending on how close the agent is to succeeding that specific task. Considering that the reward will be shaped according to the chosen subtask, learning at the low level always gets a state, takes an action and gets a reward according to its action chosen for that state.

Since the agent now has the ability to evaluate current state for each of the tasks, the decision of which task to choose becomes straightforward: the maximum one. The upper level of the hierarchy becomes selecting the maximum of the potential values for each task. After the task with maximum value is selected, the learning becomes a flat model similar to an MDP containing single task. The agent selects the primitive action. On the other hand, since the primitive action selected by the agent is towards one of the subtasks, using reward signal given by the environment can be confusing for the agent. Since we decomposed the task, the agent uses the reward of the chosen subtask. This concept can also be considered as reward shaping with task decomposition.

Formally, if we restate the idea described, the agent gets a state s in MDP M containing many tasks. The problem is decomposed into subtasks with simpler MDP M' that contains only one task. For every task i, the state s is transformed into state s'_i that belongs to MDP M'. For every task, the current situation is evaluated by collecting Q values of s'_i. At the high level, the task with maximum Q value is selected, and then the problem is treated as MDP M'. Figure 6.8 illustrates this process. From one perspective, this is a decomposition from MDP M to M' using the transformation function then evaluation of current state for each of these subtasks and selecting the subtask that gives the highest Q value in M'. We call this method as as **H**igh level **E**valuation of

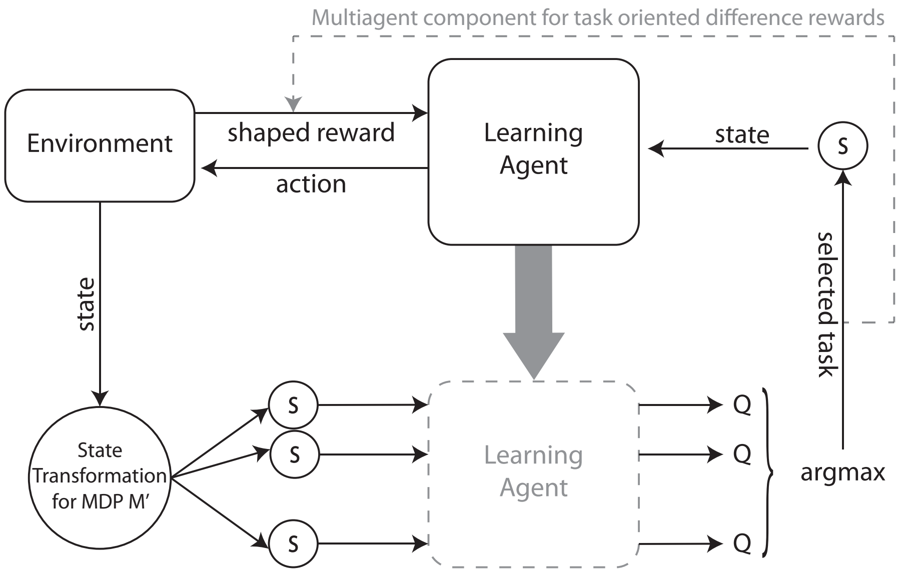


Figure 6.8: The learning diagram of HELM. The state s is transformed into states s'_i for each task i. These states are evaluated through reinforcement learner for MDP M'. The task giving the maximum value selected so that state s' is given to the agent to learn M'. In a multiagent setting, the selected task is used to shape the reward to provide the agent with difference rewards on a specific task.

Low level **MDP** using Q values (**HELM**).

Although the intuition behind the described method is a hierarchical model, the simple selection of the task with the maximum Q value prevents the model from being a true hierarchical model. The method becomes a decomposition to an one task MDP which is simpler than a hierarchical model, but still contains most of the benefits of the hierarchical models such as exploiting the repetitive nature of the problem by reusing the previous experience, simplifying the state space by decomposition and simplifying the task to learn.

One great benefit of this model is the ability to use a simpler state representation for the learning agent. Since the problem is reduced into a one task problem, the state space of the agent is reduced significantly. Instead of learning on a state representation of the complete environment, the state representation that contains information related to the specific task is enough. Another benefit of the learning model is the reuse of the knowledge. Since the agent uses the same learning for any task, at every time step,

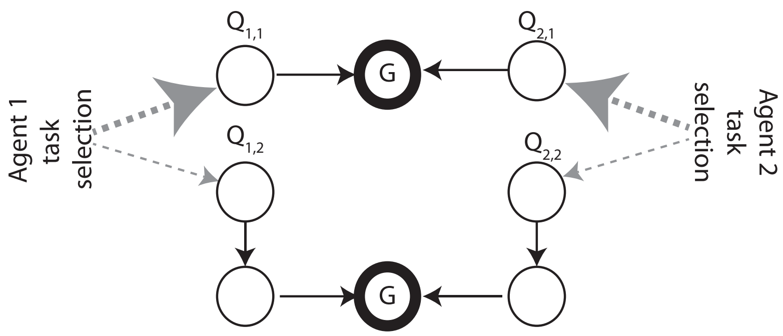


Figure 6.9: Example of two agents two goals scenario for usage of HELM. $Q_{i,j}$ represents Q value of agent i if selecting task j. It can be seen that, in the ideal case, as the closer goal state Q values will be higher, both agents will select G_1. Since one of the agents will not be able get a good reward, Q values will no longer represent closeness of the goal, which will also damage high level selection of the task in the future.

instead of learning a specific task, it learns a generic policy for all the tasks. This greatly reduces learning time of the agent.

The HELM method described in the previous section has many benefits over flat learning, but the range of the problems that it is applicable to is limited to single agent multi-task problems. On the other hand, we want to use this method in a multiagent multi-task cooperative problems. The environment contains many agents and many tasks where the agents have to learn to collaborate and share the resources.

The multiagent learning problems are harder than the single agent one in many aspects. First, the environment is highly dynamic. Every time step, the resulting state also depends on the other agents. Second, the performance of the system affecting the rewards not only depend on a single agent, it also depends on the other agents. Third, the environment becomes more crowded, and the state space gets exponentially bigger. Moreover, considering the method proposed, probably the most important difference is the nature of the task. From choosing a task and achieving a goal state, it becomes a more complex problem where the agents learn both the tasks and also to collaborate with the other agents.

If we try to apply the HELM directly to a multiagent problem, at every timestep all the agents will be choosing the best task according to the Q values of the tasks. Imagine the scenario in the previously described situation with two tasks. If we have two agents at the similar situation (Figure 6.9), looking at the Q values, they will both learn to pick the easy task. This problem not only breaks collaboration, it will also affect their performance on learning to achieve the low level task, because picking the same goal will result in unexpected rewards for the agents. Since the whole learning mechanism relies on the Q values of the low level learning, high level task selection will also fail.

Although direct application of the HELM to multiple agent setting would not work, the problem faced is nothing more than a credit assignment problem. One possible solution to this problem is to keep the decomposed subtasks multiagent. Instead of decomposing the problem into single agent single task problems, we will decompose it to multiagent single task, and provide coordination at the subtask level using reward shaping. Assuming that the agents could get a perfect reward which takes into consideration the consequences of the collaboration, it would guarantee a good reward when a low level task is achieved in a cooperative way. The consequences of such a reward will not only stabilize the Q values and bring back the proposed method to life, it will also improve it as a collaboration method.

As a reward shaping method, the Difference Rewards is a well studied and proven reward shaping method that holds the properties that HELM needs [3]. As discussed in Section 2, briefly, it gives an agent a specific reward signifying the effect of the agent on the system. Mathematically, it is defined as:

$$D_i \equiv G(z) - G(z - z_i + c_i) \; , \tag{6.13}$$

Where first term $G(z)$ is the global reward of the state z, second term $G(z - z_i + c_i)$ is the global reward of the system where the agent i is taken out and is replaced by a default action c_i. Subtraction of the second term from the first term gives the effect of the agent on the system. It is shown to perform better than G and L in many different domains [83, 38].

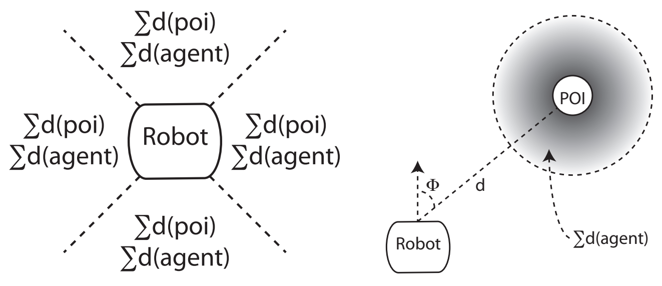
The low level MDP in HELM is a single task MDP, and the reward that the agent gets belongs to the specific task that is chosen. At this point, to be able to apply difference reward idea, the agents have to get task specific difference rewards. For that purpose, instead of using difference rewards on the whole environment, we use difference rewards for the new MDP which only contains one task. Figure 6.8 shows this additional multiagent component with a dashed arrow from the selected task to the shaped reward.

Using this information, While shaping the reward with the Equation 6.13 the function G is replaced by G_i which is global reward of the team for a given task i. We call these modified rewards "task specific difference rewards". Although the shaped reward looks like a new type of reward, it is nothing more than a difference reward for the low level MDP M' instead of MDP M.

The effects of difference rewards on HELM can be further investigated using an example scenario with two agents. Consider a case where two agents have two choices of tasks similar to the example used in single agent scenario (Figure 6.9). First, both agents decompose the problem into two tasks. The figure includes reward structures and the propagated values that affect agents' choices. As explained before, global reward results in total confusion to agent, because it also depends on other agent's effect. Not only they have problems selecting the right task, they also have problems learning to achieve a task. Local reward is a perfect signal to learn the low level tasks, but it encourages each agent to go for their own benefit even if it is not beneficial for the team. On the other hand, the difference reward distributes the reward encouraging best action for the team. There is another interesting result in this example, for the choice of task that is not beneficial for the team, the propagated value does not signify how close the agent is to the task, it results in zero. In conclusion, the propagated values signify how close the agents are to the goals but this value is non zero only for the tasks which would be beneficial for the team. Since these values discourage the agents from selecting selfish decisions, the resulting policy gives better results overall.

6.2.2 Multi-robot problem

Multi-robot problem was discussed earlier in this chapter to test interaction space (section 6.1). Multiple robots move around in an environment to maximize the observation around points of interests. To apply HELM, we represent the problem of single POI as an MDP and observing multiple POIs is a multi-task problem. For state representation, although there are many possibilities, the decision was made on previous success of the 4 quadrants (Figure 6.10-a) [38] . The state is represented with 8 numbers while 4 of them represent POI densities in 4 directions, another 4 represent rover densities in 4 directions. Discretization of the action space is done by defining 3 actions: turning left, keep straight and turn right. Same as section (section 6.1), to keep coherence with our



(a) Classical (b) Task Oriented

Figure 6.10: State Representations used for HELM. Classical State Representation consists of 8 variables representing density of Agents and POIs in 4 quadrants. Task oriented state representation consists of the angle ϕ, the distance between the robot and POI and density of agents around the POI.

locomotion research (Chapters 4 and 5) the robots have fixed rolling speed, while they can change direction. As the state space is continuous, the adaptation to RL is done by using a function approximation method called tile coding. For each variable, the range is divided into 11 tiles, and 4 different tilings. Further details about tile coding and how it works can be found in [76].

The application of HELM to the our problem is straightforward. The problem contains multiple agents that needs to observe different POIs. Here, the low level MDP is "observing a single POI with existence of other agents in the environment". This new MDP contains all the agents and only the specific POI, as opposed to all the POIs. As the task reduces to observing 1 POI, the state representation can change significantly. Here, we picked a goal oriented state representation that will make the lower level task easier to learn, which will be more precise and indirectly result in better coordination by the nature of the algorithm. For that purpose, the selected state representation is composed of 3 variables: the angle between the robot's direction and POI, the distance between the robot and the POI and robot density around the POI (Figure 6.10).

6.2.3 Experimental Results

Given a multiagent problem, using the learning schema proposed and difference rewards together is expected to learn faster and converge to a better policy. To be able to test how well HELM and difference rewards work together, we conducted experiments with various scenarios and number of agents. For every experiment, 20 statistical runs are made to show consistency of the results. For all of the experiments calculated standard errors are small enough that they are neglected on the resulting graphs.

The first experiment is on a small environment (50 by 50) with 4 agents and 4 POIs. 4 POIs are distributed to the corners and the agents are trying to observe as much as possible during episodes of 300 timesteps. The ideal behavior is having 1 robot per POI to observe. As Figure 6.11 shows, HELM combined with difference rewards (D) outperforms the rest in terms of learning speed. With a simpler task to learn, the agents have almost instant peak for convergence compared to flat learning methods. The benefit on learning speed comes from the partially hierarchical schema of the decomposition. As the information learned is reused within the low level learning, the agent basically learns the domain when it learns the subtask given by the decomposition.

On the other hand, HELM reaches a good policy only when it is combined with the difference rewards method. As explained earlier, when the low level task is not learned correctly, the theory behind using HELM on multiagent problems collapses, because global reward is not good enough even to learn the low level task. When the low level task cannot be learned high level selection is almost random, which causes more confusion to the agent. This can be clearly seen looking at the results of decomposition coupled with global rewards (G).The agents using HELM and global reward even performs worse than flat learning with G. Comparing flat learning methods with different types of rewards, the results are as expected. When the agents get shaped reward D, they outperform agents using global and local rewards.

The second experiment is relatively more complicated with a larger environment and 4 POIs concentrated at the same corner. Using this setup, we can see the effects of the more precise and goal oriented state representation that could be used after transformation to the new MDP. In this setting, the difference between HELM & D and others becomes more clear. Figure 6.12 shows that when HELM is coupled with difference rewards it outperforms both in learning time and converged policy. The difference in converged policy mainly comes from the abilities of the goal oriented state representation that is simpler and more precise than representing the whole environment (Figure 6.10-a vs Figure 6.10-b). The performance of the agents using HELM and local reward is also related to the state representation, but it will be shown in next set of experiments, local reward will not be able to keep this performance with higher number of agents. Although

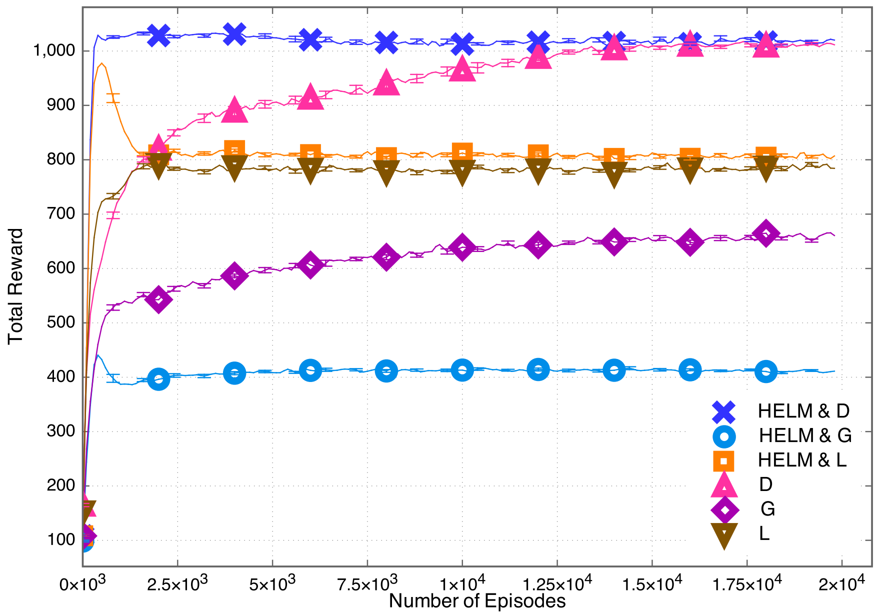


Figure 6.11: Results of HELM coupled with different reward functions on an environment containing 4 agents and 4 POIs that are spread out to different corners. D, G and L represent Difference, Global and Local Rewards respectively. The results show that HELM coupled with D has almost an instant learning experience and converges to the same point as the agents using flat learning and D

both rewards (local and difference) provide learning for the low level task, local rewards do not encourage cooperation. As the number of agents will be higher, the difference caused by the reward structures will be higher. The rest of the graph shows that coupling HELM with global reward or using flat learning with other types of rewards results in worse performance as expected.

When the number of agents is increased to 12 and 24, the Figures 6.13 and 6.14 show the consistent performance of coupling HELM with difference rewards. Again, both the converged policy and learning time is better compared to other combinations. Note that these graphs do not contain results for global reward as the performance was notably below the other four mehods.

Comparing the agents that use HELM to flat learners, it can be seen that time to converge is around $1.5 * 10^3$ episodes instead of $3 * 10^3$. From the learning time aspect,

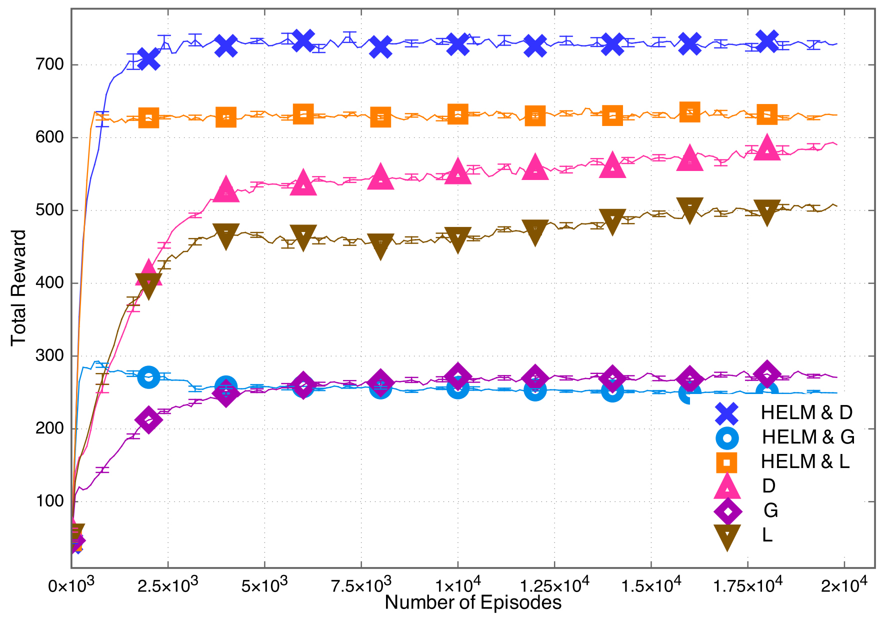


Figure 6.12: Results of HELM coupled with different reward functions on an environment containing 4 agents and 4 POIs which are concentrated in one corner. With a harder problem to learn, the difference between the methods becomes bigger. HELM & D not only converges faster, it also converges to a better policy than other methods.

the time requires to converge is less than half in all of the scenarios, and for smaller problems learning can almost be considered as instant (Figure 6.11). For a multiagent learning algorithm, this is a significant improvement.

It is shown that the converged behavior can score around 400 points higher than flat learning. Considering that flat learning scores around 2200, this gives an increase around 20% on top of the initial performance. Considering the nature of the problem, the difference can be critical in a real mars robot observation mission. We also analyzed the results from another perspective. Table 6.2 shows how many POIs are being observed at the end of each episode. Although the agents learn to increase amount of observation, it can be seen that HELM and D observes more POIs than any other combinations. Moreover, the difference is 2 POIs when there are 24 POIs and 24 agents in the system.

The experiments tested the proposed algorithm in a robot domain with different settings. With different sizes of the environment, different POI distributions, varying

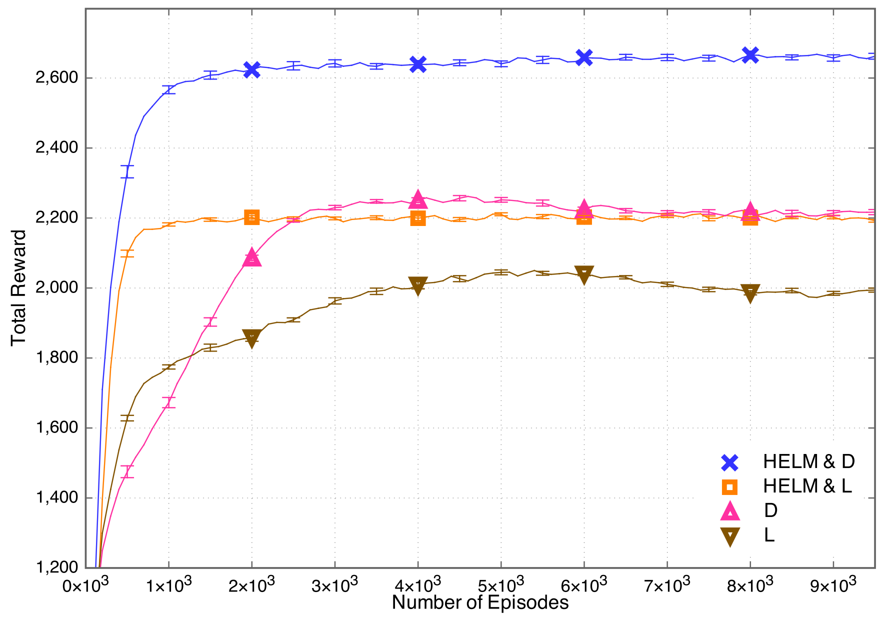


Figure 6.13: Results of HELM coupled with different reward functions on an environment containing 12 agents and 12 POIs with uniform distribution. In a crowded environment that requires more coordination, HELM & D gives an increase of 20% on overall performance of the agents

number of agents, and different reward structures, the results agreed with the claims that we made about the performance improvements. When coupled together, decomposition and task oriented difference rewards results in agents learning faster and better.

6.3 Conclusions

After establishing locomotion for a tensegrity robot in Chapters 4 and 5, through this chapter we addressed the multiagent learning for coordination of multiple robots. We concentrated on reinforcement learning and reward shaping methods. The chapter was divided into two main sections to address two different problems in reward shaping. First, we developed the reward interaction space to decrease information requirements for a shaped reward function. The results shows that we can take advantage of reward shaping methods while supplying significantly less information. The particular results

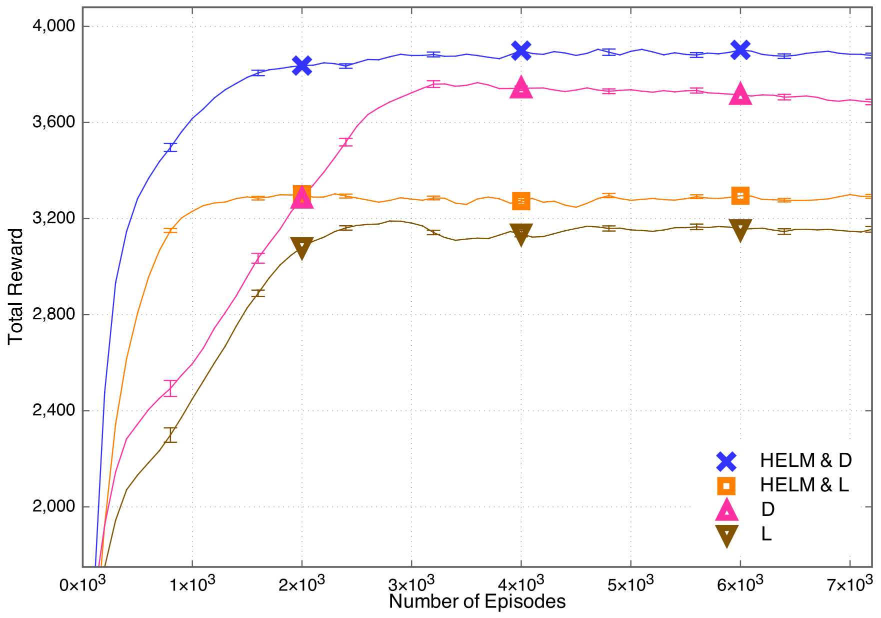


Figure 6.14: Results of HELM coupled with different reward functions on a larger environment containing 24 agents and 24 POIs that are concentrated in one area of the environment

that we had have shown that 10% of the environment can be enough to obtain the performance the same reward function with full environment.

Second aspect of multi-robot problem is about handling multi-task problems. We developed HELM, a two level decision mechism, while higher level decision is based on the lower level Q values learned using reinforcement learning. Coupled with difference rewards, HELM had success in decentralized decomposition made by the individual agents that leads to a better overall multiagent performance. Since the problem was much sim-

	12 POIs, 12 Agents			24 POIs, 24 Agents		
	D	G	L	D	G	L
HELM	11.9	9.8	11.0	20.5	13.5	18.0
Flat	11.7	11.0	11.4	18.0	15.0	16.5

Table 6.2: Number of POIs observed for different learning algorithms and rewards. There is a significant improvement when the agents use HELM and D.

pler after the decomposition, the agents were able to use a simpler but more detailed state representation. Moreover, learning was faster due to the increased simplicity of the task compared to the original multi-task problem.

Chapter 7: Tensegrity Robots and Simulation

Through the dissertation, simulation of tensegrity robots plays a critical role for multiagent learning. First, simulations were necessary to iterate between design of a novel robot and designing a locomotion algorithm for the same robot. Simulations allowed us to be flexible and try different configurations of hardware, to decide the requirements for our next prototype. Second, simulations speed up experimentation with different policies. Multiagent learning can take thousands of episodes to learn a policy. Considering the time that it takes to setup experiments with a hardware, simulations are necessary.

Additionally, experimenting on a hardware can be costly. Since learning algorithms try every possible control policy, breaking the robot is probable and costly. During the learning of locomotion in Chapter 5, we also observed that the algorithm was trying non-feasible policies that we were penalizing for breaking the robot.

Despite these benefits, working with a simulator has its own challenges. Modeling the robot and the dynamics are critical. Most of the time, transferring the learned policies in a simulator to a hardware can be challenging due to the differences between the real world and the simulation environment. In this section, we discuss the challenges of simulating a tensegrity robot. We present NASA Tensegrity Robotics Toolkit, and how we validated it using a previously built tensegrity robot, ReCTeR, by Caluwaerts et al [15].

The validation of NTRT is done using the existing robot, ReCTeR. The limited capabilities of ReCTeR led to an incrementally superior icosahedron robot: SUPERBall. Through the Chapters 4 and 5, we used SUPERBall model to learn rolling locomotion and analyzed the requirements of the algorithm. Although we did not design or build SUPERBall hardware, the results of the locomotion algorithms that we presented directly contributed to the design of the current prototype. In this chapter, we will present the design and specifications of the SUPERBall and discuss the role that NTRT and our results played during the design process.

7.1 NTRT

Simulation of Tensegrity robots is critical for application of intelligent controls. Simulation provides multiple advantages, such as being able to train controllers faster than training them on actual hardware, or using the simulator to explore design options and hardware requirements. In our paper, we use the (soon to be Open Source) NTRT simulator that has been developed on top of the Open-Source Bullet Physics Engine. The Bullet Physics Engine is a discrete timestep, iterative physics simulator that handles collisions and interactions between different types of objects [11].

The NTRT simulator relies on the Bullet Physics Engine to handle movement and collisions of rigid-body objects. Since game physics require real time simulation, the advantage of using Bullet is to be able to handle collisions without excessive processing power. On the other hand, to simulate the tensional elements, we needed precise elastic components whose rest lengths can be actively controlled to simulate controls of the robot. For this purpose, we implemented our own tensional elements that apply tension according to their stretch. Similar to previous chapters, we use the term 'muscle' to present these tensional elements. We added required dynamics to simulate these muscles in bullet physics engine.

A muscle is attached to two different rigid bodies from specific points. We assume that muscles are abstract elements that apply force to these two rigid bodies according to their tensions. They have basic properties, such as rest length (length without stretch) and elasticity coefficient. The tension of a muscle is computed by looking at the current distance between two attachment points and the rest length of the muscle. The intensity of the force vector exerted to the bodies is defined as:

$$|F| = (l - r) * c$$

where l is the length of the cable defined as the distance between the 2 anchor points for the cable. r is the rest length of the part of the cable that lies between two anchors. c is the elasticity coefficient of the cable that is used.

For simulation purposes, we assume that the muscles have negligible weight compared to the rest of the structure. Moreover, we also ignore the interactions of the muscles with the rest of the environment. These two assumptions are understandable because the tensional elements stay inside the structure and do not interact with other objects except for extreme deformations. Also, the weight of the physical cables is extremely low compared to the weight of the rest of the robot.

To simulate the actuators that change the lengths of the cables, we use a direct approach that changes rest lengths of the cables connected to the actuators.

$$r_{t+1} = r_t \pm \delta t * s_m$$

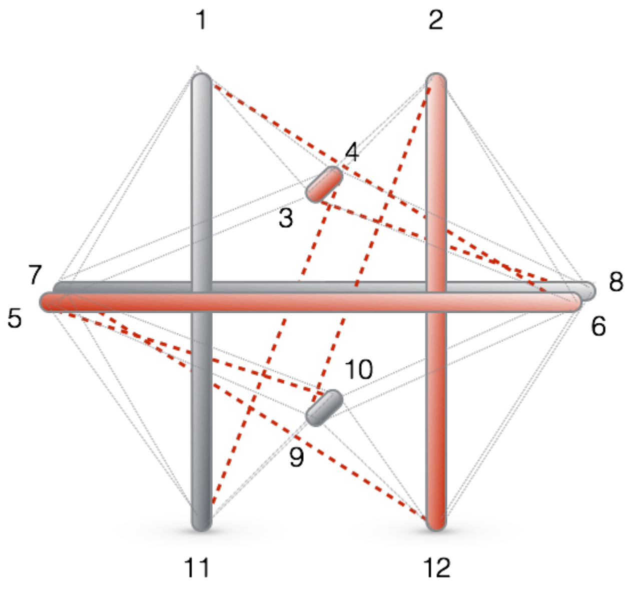
Where r_t is the rest length of the cable at timestep t, δt is the time between two timesteps of the simulator and s_m is the motor speed of the actuators. The motors have constant speed (0.2 m/s) while pulling or releasing a cable. The motors cannot handle tensions that are higher than a given threshold (150 N). When the tension is higher than the threshold, we assume that the motors cannot pull or hold the cable. In such scenarios, the cable is released independent of the intention of the controller.

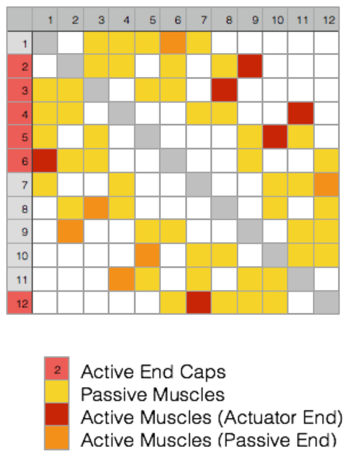
7.2 ReCTeR

To be able to judge the performance of the approach taken with bullet physics engine and our own muscle models, we will use a physical tensegrity robot, ReCTeR. In this section, we first present physical properties of ReCTeR, then analyze the comparison between the physical robot and its model in our simulation environment.

ReCTeR (Reservoir Compliant Tensegrity Robot) was built by Ken Caluwaerts to study compliant locomotion with tensegrity structures and to validate our simulation results [15, 14]. The robot is a lightweight, underactuated tensegrity prototype with rich sensor integration, based on off-the-shelf components (Figure 7.6). In the following paragraphs, we detail the mechanical and sensor design of ReCTeR.

As opposed to the model that we use in this dissertation, the 24 shell tensile elements of ReCTeR are passive, the robot can move, fold and deform by six additional actuated springs running through the assembly. The connection pattern is shown in Figure 7.1. ReCTeR has a total mass of $1.1kg$ (batteries included), which is achieved by using carbon fiber struts ($8mm$ outer diameter). The tensegrity principle allows to make effective use of the axial strength of the carbon fiber. In his dissertation, Caluwaerts shows that ReCTeR can survive drop tests of $0.5m$ and various experiments without any of the struts splicing or breaking, clearly demonstrating how tensegrity structures use





	1	2	3	4	5	6	7	8	9	10	11	12
1	▨		🟨	🟨	🟨	🟧	🟨					
2		▨	🟨	🟨		🟨		🟨	🟥			
3	🟨	🟨	▨		🟨	🟨		🟥				
4	🟨	🟨		▨			🟨	🟨			🟥	
5	🟨		🟨		▨				🟨	🟥	🟨	
6	🟥	🟨	🟨			▨			🟨			🟨
7	🟨			🟨			▨			🟨	🟨	🟧
8		🟨	🟧	🟨				▨		🟨		🟨
9		🟧			🟨	🟨			▨		🟨	🟨
10					🟧		🟨	🟨		▨	🟨	🟨
11				🟧	🟨		🟨		🟨	🟨	▨	
12						🟨	🟥	🟨	🟨	🟨		▨

2	Active End Caps
🟨	Passive Muscles
🟥	Active Muscles (Actuator End)
🟧	Active Muscles (Passive End)

Figure 7.1: The connection pattern of ReCTeR robot. 3 out of 6 struts are actuated. There are 6 active muscles in addition to 24 passive muscles composing icosahedron skeleton.

structural elements in pure compression or tension [14].

Three of ReCTeR's struts are actuated (two actuators each), while the other three are fully passive and sensorless (see Figure7.1). The total mass of the struts is $0.05kg$ and $0.270kg$ for the passive and active struts respectively. The six actuated springs are selected such that each end cap has exactly one actuated spring attached to it. Detailed design information about ReCTeR can be found in previous articles by Caluwaerts [14] and Bruce et al [10]

While ReCTeR shows basic capabilities of a tensegrity robot such as folding, compliance and deformation, it is actuated using additional 6 muscles instead of the outer shell. This limits the capabilities in terms of locomotion and symmetry. Moreover, with the goal of being lightweight, the robot is designed using motors that cannot apply high torque that might be necessary for continuous rolling. Using the lessons learned with ReCTeR, the necessity of a stronger robot emerged the design of the SUPERball that is described in Section 7.4.

7.3 Validating NTRT using ReCTeR

To validate the results that we obtain using NTRT, we studied the gap between a real tensegrity robot and our simulation environment. We used ReCTeR described in Section 7.2, modeled in NTRT (Section 7.1) and observed the difference of behavior between the real robot and its model in NTRT. The same control commands are given to both physical and simulation environments. To track the full state of the robot, we used an active marker motion capture setup. Each passive strut was fitted with 2 markers, while each active strut had 3 markers.

First, we start with a basic experimental setup using a single strut connected to the floor and ceiling by four springs, two of which were actuated. This setup is shown in Figure 7.2. To get a precise comparison between simulators and hardware, we actuated the strut with different control policies and compared the resulting behavior in Fig. 7.3. We also compared the results with Euler-Lagrange solver. As opposed to NTRT, Euler-Lagrange solver does not simulate based on discrete iteration, it solves the dynamics equations for given structure. It is physically more accurate, but it cannot handle ground interactions as well as NTRT. Since this experimental setup does not have any ground interactions, it is more accurate. The results show that EL solver, NTRT and physical robot all have matching results, although the NTRT simulator was more damped than the physical setup.

Next, the six strut tensegrity was placed on one of its triangular faces and two of the top springs were actuated, as shown in Fig. 7.4b. We tracked the displacement of one of the nodes that is in air. The incident strut was suspended in the air by a total of 10 springs. The robot was controlled using inverse kinematics to draw the letter 'a' using the end of the strut. This setup is shown in Figure 7.4a. The resulting displacements in simulation and hardware are shown in Figure 7.4b. The results overlap completely. The results also match with Euler-Lagrange simulator that is physically more accurate.

Next, we compared NTRT simulator to ReCTeR hardware in a dynamic scenario. The goal of the experiment is to verify that the simulator can replicate ground interactions and can simulate the conversion of potential energy into kinematic energy when a spring is released. The experimental setup is shown in Fig. 7.5. The robot initially has a complex, non-minimal ground contact (four nodes on the ground) and three of the actuated springs are tensioned. Next, one tensioned actuated spring is unwound, which

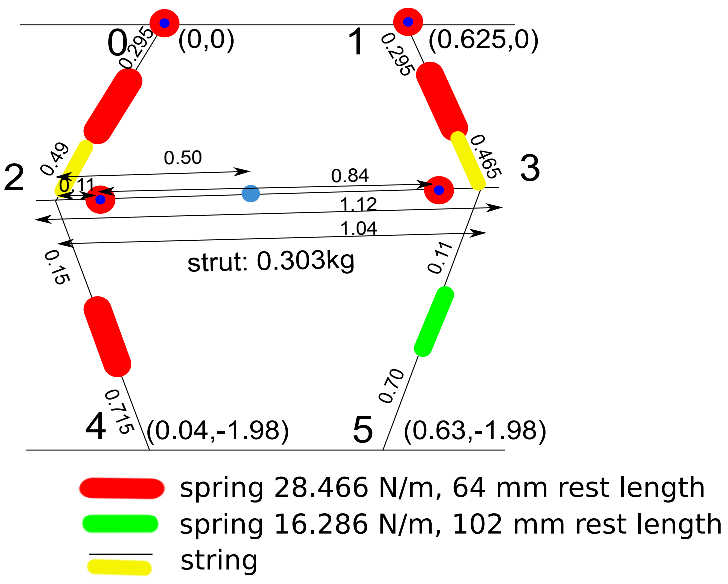
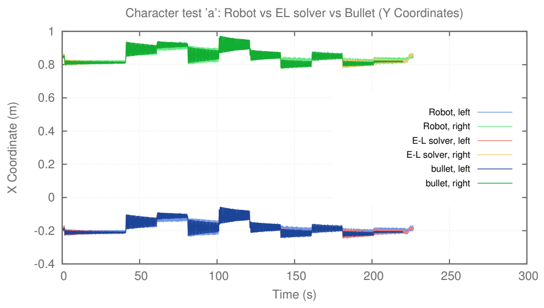
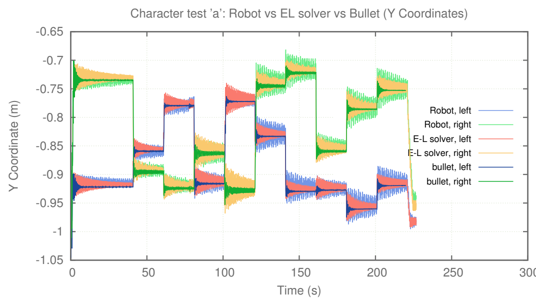


Figure 7.2: The experimental setup for one of the struts. The strut is attached from 4 points, with 3 passive and 1 active muscles.

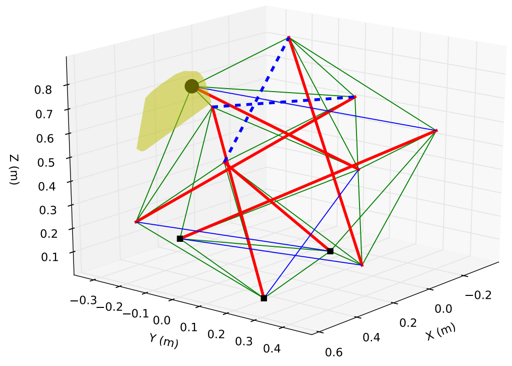


(a) x coordinate

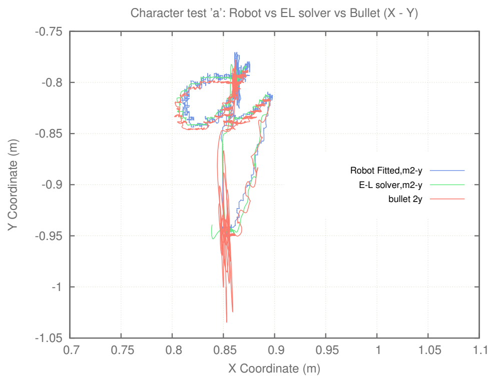


(b) y coordinate

Figure 7.3: The comparison of the trajectories of the end of the rod in motion capture data, NTRT simulations and EL-solver.



(a) Experimental setup



(b) Results

Figure 7.4: The comparison of the trajectory of the specific end of the robot in motion capture data vs EL-solver and NTRT. The letter ‘a’ is drawn using inverse kinematics and the results precisely match in all three environments.

causes the robot to roll over.

As the experiment also depends on the initial state of the robot, we copied the observed initial state from the motion capture data to the NTRT simulator and allowed the robot to reach the static equilibrium predicted by the simulator (instead of the observed one). The recorded motor positions were then fed into the simulator.

The experimental setup is shown in Figure 7.5. One of the tensioned springs is released ($32cm$ to $53.5cm$ at $0.6m/s$), causing the robot to perform a flop. Comparing the simulation and motion capture data, we observed a time averaged error of the nodes' vertical positions of less than 5% of the robot's diameter for all nodes. The robot sometimes fails to roll on slippery terrains, which we also observed when reducing the NTRT friction parameters.

Overall, the results show that NTRT can simulate ReCTeR with a small error margin. We tested the simulation of actuators using one strut, the internal dynamics of the robot when it is stationary, and the ground interactions with a flopping scenario. The ultimate comparison between the two systems would be comparison of locomotion, but considering that ReCTeR is underactuated, it is not possible to obtain continuous rolling behavior.

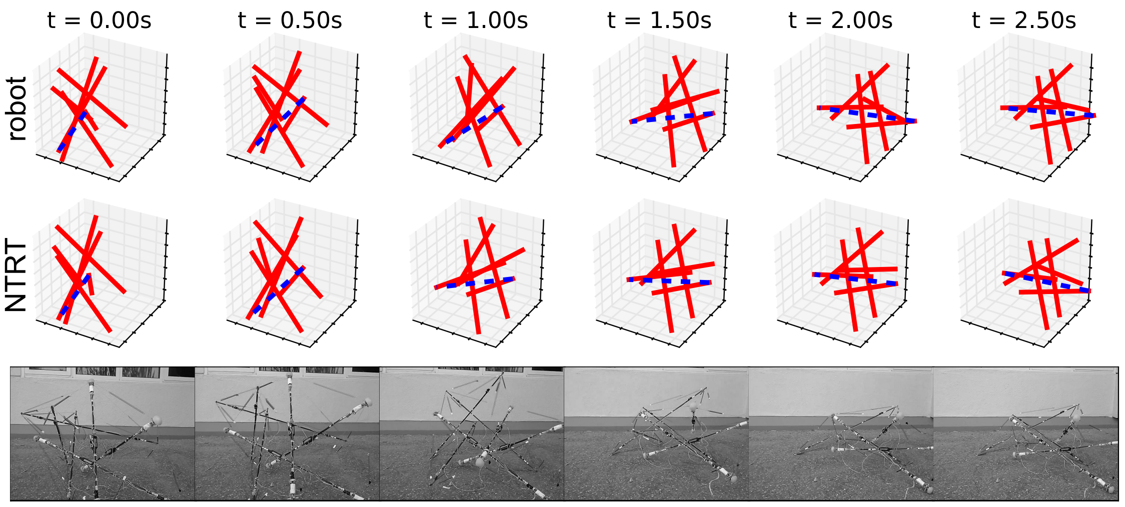


Figure 7.5: Comparing robot and NTRT dynamics. The tensioned spring indicated by the dashed line is released to cause the robot to perform a single flop.

7.4 SUPERball

In previous sections, we presented NTRT and presented its realism by comparing to physical robot ReCTeR. ReCTeR has limited capabilities with its underactuated skeleton. Rolling locomotion for a tensegrity robot has higher requirements in terms of hardware. To match these requirements, the results that we presented in Chapters 4 and 5 emerged to a new design by Bruce et al: SUPERBall [10].

The Spherical Underactuated Planetary Exploration Robot (SUPERball) is a tensegrity icosahedron robot currently under development at the NASA Ames Research Center. The main design goal of SUPERball is to be a more capable robot than a prior prototype ReCTeR, to provide more reliable sensors, and to handle rougher environments. The difference between the two robots is illustrated in Figure 7.6, where ReCTeR and one of the struts of the SUPERBall are shown side by side.

ReCTeR's actuation is limited to six additional tensile elements on top of the passive icosahedron skeleton of 24 passive muscles. The six actuated members connect opposite sides of the shell. Hence, only 20% of ReCTeRs tensile members are actuated. For SUPERball, the goal is to have the possibility to actuate all tensile elements. The current prototype is going to have 12 out of 24 actuated tensile elements, but considering our

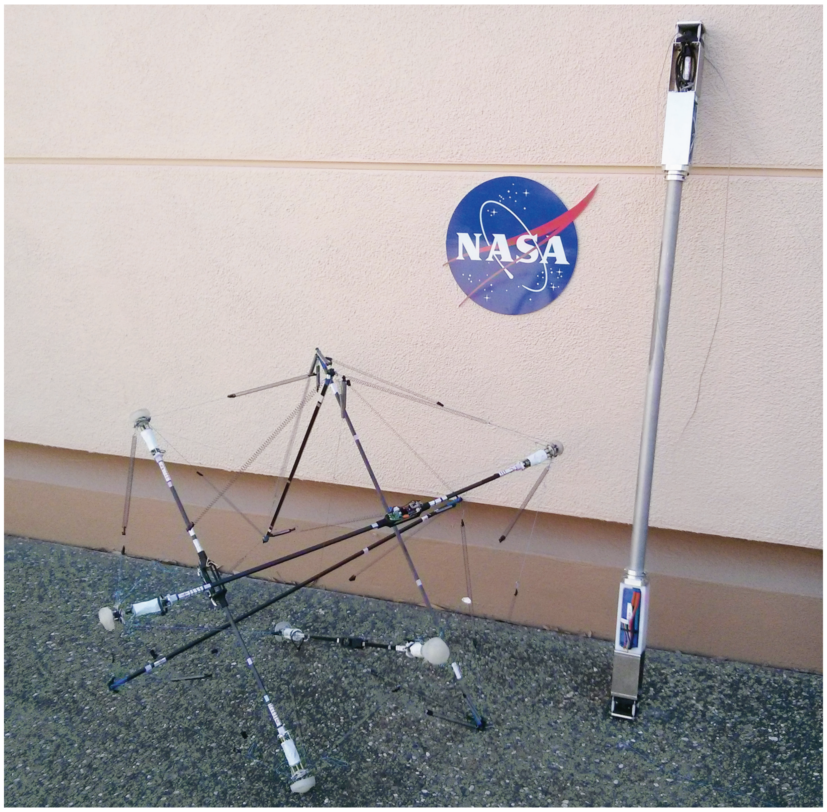


Figure 7.6: ReCTeR (left), and one strut of the modular SUPERBall (right) that is under development at NASA Ames Research Center.

future goal, this dissertation studies the fully actuated robot. The main rationale for full actuation is that we aim to explore full capabilities of non-limited tensegrity robots.

The comparison of specifications between ReCTeR and SUPERBall is shown in Table 7.1. With a total mass of $1.1kg$ (batteries included), ReCTeR cannot carry any scientific payload without a significant impact on the robot's dynamics. This is another reason to develop a new platform. SUPERball's mass with 50% actuated tensile members is around $15kg$. This larger mass will allow us to explore the behavior of tensegrity robots with a small payload suspended by tensile elements in the center of the robot [75]. Another important improvement is SUPERball's high power-to-weight ratio. ReCTeR's small brushed Direct Current (DC) motors are often maxed out ($25W/kg$), while SUPERball has significant headroom ($> 80W/kg$, final design $> 100W/kg$). This allows for dynamic

Table 7.1: SUPERball Design Requirements

	l_{strut}	Δl	$k_{passive}$	Ctrl. freq.	$\max \tau$
ReCTeR	1m	0.3m/s	28.4N/m	40Hz	0.03Nm
SUPERball	1.5m	0.26m/s	500N/m	100Hz	3Nm

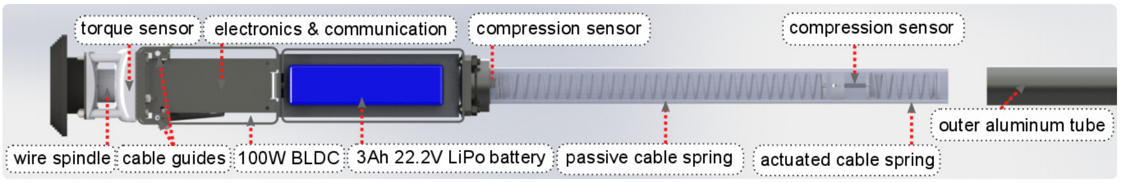


Figure 7.7: Cross-section of the end of the rod for the current design of the SUPERBall.

motion even in energetically suboptimal regimes.

Typical spring tensions are $5N$ to $20N$ for ReCTeR, while the average tension in SUPERball's current configuration is $50N$. The SUPERball hardware is designed to handle tensile forces up to $500N$. This specifications fit the locomotion algorithms that we presented in Chapters 4 and 5. One interesting aspect of the tensegrity icosahedron configuration is that the design can conveniently be scaled up or down. The design can easily be changed to have a bigger robot. Forces scale approximately linear as a function of the robot's mass. For practical purposes, SUPERball is deployed in its minimum diameter configuration where each strut is $1.5m$ in length.

The scalability of the robot can be seen with an example of larger struts. Imagine the same robot with 4.5 meter diameter struts instead of 1.5 meter. Each strut becomes 3 times longer. In its default configuration, the robot becomes approximately 9 times larger in volume. Increasing the diameter of the robot does not significantly increase its total mass, because lightweight hollow aluminum tubes are used to connect the end caps. More precisely, SUPERBall has $35mm$ diameter tubes with a $1.25mm$ wall thickness. These tubes currently account for 15% of the robot's mass. Tripling the robot's diameter ($4.5m$ struts) with the same end caps would only increase each strut's mass 26%. Since the end caps handle higher tensions than the current locomotion algorithm requires, the claims about the locomotion and the design would also work for a larger version of

SUPERBall.

Another advantage of SUPERball is its highly modular platform. This allows us to explore locomotion strategies for tensegrity robots with a shape other than icosahedron. These strut's basic elements are the end cap, with each end cap being independent. Every end cap provides actuation, battery power, various sensors, and wireless communication. In the current design, each SUPERball end cap houses a single $100W$ brushless DC motor and approximately $70Wh$ of battery power. Thus, we can currently control twelve tensile members in the icosahedron configuration (50%). The sensory equipment of each end cap consists of two tensile force transducers, a torque sensor on the motor, and an inertial measurement unit with nine degrees of freedom.

The current design has only 12 muscles actuated out of 24. On the other hand, the other 12 passive muscles are also designed to provide compliance. The cables are coupled with compression springs inside the tube. The cable runs from the opposing end cap to the spring end cap, passes through a cable housing assembly inside the spring end cap, and then connects to a compression spring located within the spring end cap's tube. Although not actively controlled, these compression springs are easily replaceable to change the overall stiffness structure.

From simulation perspective, using compression springs inside the tube provides significant advantages. In ReCTeR, compliance is provided by tensional springs that are attached in the middle of the muscles. Since cables are significantly lighter than these springs, the springs compose the big part of the weight of the muscles. In previous section, we explained that we ignore the weights of the muscles in NTRT. In SUPERBall, placing the springs inside the tubes supports this assumption by decreasing the weights of the muscles even further. Moreover, in ReCTeR, it is possible for the attached springs to cause motion that is perpendicular to each muscle. These vibrations can change the behavior of the robot during locomotion. SUPERBall design removes this possibility by placing the springs into the tubes.

Overall, the SUPERBall robot will be the first untethered icosahedron robot with 12 actuated muscles. These properties with is sensory and actuation capabilities allow it to be the first robot for continuous rolling tensegrity locomotion. With the compliant design with springs located in tubes, it is a perfect match for the simulation assumptions made in NTRT. As a result, once the SUPERBall is completed, the transfer of the learned policies in Chapters 4 and 5 is going to be possible and straightforward.

Chapter 8: Conclusion

The work presented in this thesis brings together the topics of multiagent learning and tensegrity robotics. Tensegrity Robotics is a fairly new research topic that offers the advantages of tensegrity structures to the robotics. These properties such as being compliant, lightweight, robust and distributed bring significant advantages to play a big role in the future of mobile robots. On the other hand, these properties come with the cost of being hard to control due to nonlinear interactions and oscillatory nature. To address these challenges, we developed and used multiagent learning methods.

We addressed two problems in tensegrity robotics: rolling locomotion and multi-robot coordination. While doing so, we used two different multiagent learning methods: Reinforcement learning and Evolutionary Algorithms. In multiagent case, both methods contain the credit assignment problem. In addition to the application of these methods to tensegrity robotics problems, we developed 3 methods that improves the current state of the art on the credit assignment problem: historical average, interaction space and HELM.

8.1 Contributions

The contributions of this thesis can be classified into two different perspectives: multi-agent learning methods and tensegrity robotics perspective. From multiagent learning perspective, we have four main contributions:

1.1 We developed historical average fitness shaping to favor robust solutions for co-evolutionary algorithms. As a result, we obtained faster learning and more robust locomotion for the tensegrity robot.

1.2 We introduced pooling in coevolutionary algorithms for knowledge reuse. This provides faster learning and smaller search space allowing us to learn directional rolling locomotion for the tensegrity robot.

1.3 We derived the concept of interaction space to measure the required information for a shaped reward in reinforcement learning. As a result, we shrunk the information used by reward shaping function while keeping the same performance from learning.

1.4 We provided HELM as a distributed task allocation method for multiagent reinforcement learning. As a result, multiple agents can coordinate between tasks using the reward shaping at a lower level single task learning.

From tensegrity robotics perspective, the consequences of these methods are entirely different. The general contribution is the approach that we take to the chicken-and-egg problem of designing locomotion algorithm and designing the robot that can handle the locomotion algorithm. In general, the literature contains research on designing locomotion algorithms for a given hardware or research on designing a new hardware for a form of locomotion. In our case, we have both problems dependent on each other. For such a problem, we take advantage of the simulation environment that we develop. We start with an initial configuration and a simple locomotion approach, according to our feasibility results, we decide on the further specifications of the hardware for the next prototype. Then we use a more advanced locomotion approach using the specifications of the hardware. In addition to this workflow, the dissertation contains different types of contributions to tensegrity robotics:

2.1 We derived the first rolling locomotion by body deformation for a tensegrity robot. Due to the complexity of the task, classical control methods does not provide a solution to the problem of rolling locomotion for tensegrity robots. The success that we obtain by using coevolutionary algorithms provides a roadmap to designing locomotion algorithms for complex tensegrity robots. The designed control system is open loop and distributed, making it easy to deploy for a given hardware.

2.2 We used a model of an actual tensegrity robot that is under production, and test the simulator by comparing it with a physical icosahedron robot. We analyzed the resulting behavior of using open loop controls for rolling by looking at the tensions, power and deformations. The results show the feasibility of the concept of tensegrity robots rolling using body deformations.

2.3 With flop and roll, we designed a closed loop rolling algorithm using minimal amount of sensory input. We provide the first directional and continuous rolling locomotion for a tensegrity robot. We first analyze the locomotion in terms of internal characteristics such as tension, length, power of the muscles to show the feasibility. Second, we analyze the observable behavior such as path, footprints and displacements of the end of the rods. As a result, we show the similarities between rolling tensegrity locomotion and legged locomotion, as well as additional advantages of rolling locomotion such as compliance and robustness to balancing problem. By testing the locomotion against different external conditions, we show the robustness of the learned behavior against noise and external forces.

2.4 We presented a tensegrity simulator that is both realistic but also fast enough to use for evolutionary algorithms. Evolutionary algorithms require simulation of the same scenario thousands of times. If the simulator is not fast enough, learning based controls is not an option anymore. We overcome this by using bullet physics engine with discrete time step and removing unnecessary calculations caused by elastic body collisions and dynamics. As a result of our comparisons with a physical tensegrity robot, we conclude that the simulation environment does not cause significant amount of difference.

8.2 Future work

The dissertation contains many future research directions. Although each chapter has different future work in different disciplines, the main future direction is the application of these experiments to the physical SUPERBall after its development. The gap between the physical robots and simulation environments is a known problem in the robotics. In this dissertation, we tightened this gap by testing NTRT with ReCTeR for specific tasks. Moreover, we tested our algorithms against different types of noises that can exists in a physical environment. Despite these efforts, transferring the given algorithms to the hardware will bring additional challenges. Successful application of these algorithms and their comparison with simulation environments is the next step for our work.

In addition to discussed hardware application, each chapter contains its own open ended questions:

- In Chapter 3, we present historical average, and discuss its advantages. The algorithm is applied to tensegrity locomotion problem to provide more robust results.

Emprical results show the success of the algorithm in this domain, but the analysis of the algorithm in the context of evolutionary game theory is left as a promising future work. An analysis on the area of the solution space that is explored by historical average can provide bounds for the success of the algorithm in different domains.

- Chapter 4 contains open loop rolling locomotion using 24 muscles. During the learning, the optimized criteria is the distance traveled by the robot. On the other hand, lowering the energy used is a critical problem untethered robots. To reduce the energy cost, there are two research directions to be investigated. First direction is multi-objective optimization using distance and energy used. Second direction is obtaining rolling locomotion using less active muscles. We believe that actuating 8 muscles (out of 24) can be enough to obtain rolling for an icosahedron robot. Decreasing number of necessary actuators can decrease both energy cost as well as the weight of the robot.

- Chapter 5 presents flop and roll as a closed loop locomotion. Reducing the energy cost and the number of actuators is the definite future direction. Another research direction is formalizing the algorithm for more complex tensegrity structures.

- Chapter 6 presents both interaction space and HELM. Interaction space finds minimum information required to keep the same performance in reward shaping. It is possible to extend this work towards finding the amount of information required to guarantee a performance bound. While defining HELM, we empirically show the necessity of difference rewards to provide high level collaboration. In our opinion, the best possible way to extend this work is to provide theoretical proof for convergence to stable policies.

- Chapter 7 discusses simulation of tensegrity robots and validation using a hardware. NTRT is open to progress in terms of simulating collision with ropes. The next step for the validation of NTRT is studying comparison of a continuously rolling SUPERBall vs its simulated model.

In this dissertation we showed the capabilities of multiagent learning methods by addressing problems of a fairly new research field in robotics. The results and the open research directions that we provided show the capabilities of tensegrity robots as well as the success of multiagent learning methods on complex problems. As a consequence, this dissertation shows a roadmap to control, coordination, design and simulation of rolling compliant robots that promise many advantages to the field of robotics.

Bibliography

[1] A. Agogino and K. Tumer. Multi agent reward analysis for learning in noisy domains. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Utrecht, Netherlands, July 2005.

[2] A. K. Agogino and K. Tumer. Handling communication restrictions and team formation in congestion games. *Journal of Autonomous Agents and Multi-Agent Systems*, 13(1):97–115, 2006.

[3] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[4] Adrian K. Agogino and Kagan Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17:320–338, October 2008.

[5] W.B. Arthur. Inductive reasoning and bounded rationality. *The American economic review*, 84(2):406–411, 1994.

[6] Tim D Barfoot, Ernest JP Earon, and Gabriele MT D'Eleuterio. Experiments in learning distributed control for a hexapod robot. *Robotics and Autonomous Systems*, 54(10):864–872, 2006.

[7] N. Bel Hadj Ali, L. Rhode-Barbarigos, A.A. Pascual Albi, and I.F.C. Smith. Design optimization and dynamic analysis of a tensegrity-based footbridge. *Engineering Structures*, 32(11):3650–3659, 2010.

[8] Shourov Bhattacharya and Sunil K Agrawal. Spherical rolling robot: A design and motion planning studies. *Robotics and Automation, IEEE Transactions on*, 16(6):835–839, 2000.

[9] Antonio Bicchi, Andrea Balluchi, Domenico Prattichizzo, and Andrea Gorelli. Introducing the "sphericle": an experimental testbed for research and teaching in nonholonomy. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2620–2625. IEEE, 1997.

[10] Jonathan Bruce, Ken Caluwaerts, Atil Iscen, Andrew Sabelhaus, and Vytas SunSpiral. Design and evolution of a modular tensegrity robot platform. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[11] BulletPhysicsEngine. http://www.bulletphysics.org/, 2013.

[12] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.

[13] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC PressI Llc, 2010.

[14] Ken Caluwaerts. *Design and Computational Aspects of Compliant Tensegrity Robots*. PhD thesis, Ghent University, 2014.

[15] Ken Caluwaerts, Michiel D'Haene, David Verstraeten, and Benjamin Schrauwen. Locomotion without a brain: physical reservoir computing in tensegrity structures. *Artificial Life*, 19(1):35–66, 2013.

[16] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[17] Mitchell Colby and Kagan Tumer. Shaping fitness functions for coevolving cooperative multiagent systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 425–432. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[18] Kenneth A De Jong. *Evolutionary computation: a unified approach.* MIT press, 2006.

[19] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 225–232. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[20] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. Artificial intelligence through simulated evolution. 1966.

[21] M. Fujiia, S. Yoshiia, and Y. Kakazub. Movement control of tensegrity robot. *Intelligent Autonomous Systems 9: IAS-9*, 9:290, 2006.

[22] Buckminster Fuller. Tensegrity. *Portfolio and Art News Annual*, 4:112–127, 1961.

[23] David Edward Goldberg et al. *Genetic algorithms in search, optimization, and machine learning*, volume 412. Addison-wesley Reading Menlo Park, 1989.

[24] C.V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22(1):143–174, 2004.

[25] S. Hirai and R. Imuta. Dynamic simulation of six-strut tensegrity robot rolling. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 198–204, Dec 2012.

[26] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, June 1998.

[27] D E Ingber. Tensegrity I. Cell structure and hierarchical systems biology. *Journal of Cell Science*, 116(7):1157–1173, 2003.

[28] Donald E. Ingber. Cellular tensegrity: defining new rules of biologic design that govern cytoskeleton. *Journal of Cell Science*, 104:613–627, 1993.

[29] Donald E Ingber. Tensegrity: The architectural basis of cellular mechanotransduction. *Annual Review of Physiology*, 59(1):575–599, 1997.

[30] Atil Iscen, Adrian Agogino, Vytas SunSpiral, and Kagan Tumer. Learning to control complex tensegrity robots. In *AAMAS*, 2013.

[31] Atil Iscen, Adrian Agogino, Vytas SunSpiral, and Kagan Tumer. Robust distributed control of rolling tensegrity robot. In *The Autonomous Robots and Multirobot Systems (ARMS) workshop at AAMAS 2013*, 2013.

[32] Atil Iscen, Adrian K. Agogino, Vytas SunSpiral, and Kagan Tumer. Controlling tensegrity robots through evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1293–1300, 2013.

[33] Atil Iscen and Kagan Tumer. Decentralized coordination via task decomposition and reward shaping. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1269–1270. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[34] Sergi Hernàndez Juan and Josep M. Mirats Tur. Tensegrity frameworks: Static analysis review. *Mechanism and Machine Theory*, 43(7):859 – 881, 2008.

[35] L.P. Kaelbling, M.L. Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[36] H. Klimke and S. Stephan. The making of a tensegrity tower. In *IASS Symposium*, Montpellier, 2004.

[37] M. Knudson and K. Tumer. Coevolution of heterogeneous multi-robot teams. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 127–134. ACM, 2010.

[38] M. Knudson and K. Tumer. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems*, 59(6):410–420, 2011.

[39] Yuusuke Koizumi, Mizuho Shibata, and Shinichi Hirai. Rolling tensegrity driven by pneumatic soft actuators. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1988–1993. IEEE, 2012.

[40] J.R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *The Journal of Machine Learning Research*, 7:1789–1828, 2006.

[41] K Koohestani. Form-finding of tensegrity structures via genetic algorithm. *International Journal of Solids and Structures*, 49(5):739–747, 2012.

[42] A.D. Laud. *Theory and application of reward shaping in reinforcement learning.* PhD thesis, University of Illinois, 2004.

[43] Stephen Levin. The tensegrity-truss as a model for spine mechanics: Biotensegrity. *Journal of Mechanics in Medicine and Biology*, 2:375–388, 2002.

[44] Yue Li, Xi-Qiao Feng, Yan-Ping Cao, and Huajian Gao. A monte carlo form-finding method for large scale regular and irregular tensegrity structures. *International Journal of Solids and Structures*, 47(14):1888–1898, 2010.

[45] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In Zoubin Ghahramani, editor, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 601–608. ACM, 2007.

[46] M. Masic and et al. Algebraic tensegrity form-finding. *International Journal of Solids and Structures*, 42:4833–4858, 2005.

[47] Milenko Masic, Robert E Skelton, and Philip E Gill. Algebraic tensegrity form-finding. *International Journal of Solids and Structures*, 42(16):4833–4858, 2005.

[48] Manipulative Medicine, Lumbar Spine, Toronto June, The Bulletin, and Structural Integration. Continuous Tension , Discontinuous Compression : A Model for Biomechanical Support of the Body. *Spine*, pages 4–7, 2006.

[49] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287, San Francisco, CA, USA, 1999.

[50] O. Orki, A. Ayali, O. Shai, and U. Ben-Hanan. Modeling of caterpillar crawl using novel tensegrity structures. *Bioinspiration & Biomimetics*, 7(4):046006, 2012.

[51] O. Orki, O. Shai, A. Ayali, and U. Ben-Hanan. A Model of Caterpillar Locomotion Based on Assur Tensegrity Structures. *Proceedings of the ASME 2011 IDETC/CIE*, August 2011.

[52] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[53] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and MultiAgent Systems*, 11(3):387–434, 2005.

[54] Liviu Panait, Keith Sullivan, and Sean Luke. Lenient learners in cooperative multiagent systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 801–803. ACM, 2006.

[55] Liviu Panait, Karl Tuyls, and Sean Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *The Journal of Machine Learning Research*, 9:423–457, 2008.

[56] C. Paul, J.W. Roberts, H. Lipson, and F.J.V. Cuevas. Gait production in a tensegrity based robot. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pages 216–222. IEEE, 2005.

[57] C. Paul, F.J. Valero-Cuevas, and H. Lipson. Design and control of tensegrity robots for locomotion. *Robotics, IEEE Transactions on*, 22(5):944–957, 2006.

[58] Chandana Paul, Hod Lipson, and Francisco J. Valero Cuevas. Evolutionary form-finding of tensegrity structures. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 3–10, New York, NY, USA, 2005. ACM.

[59] Chandana Paul, Hod Lipson, and Francisco J Valero Cuevas. Evolutionary form-finding of tensegrity structures. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 3–10. ACM, 2005.

[60] Mitchell A Potter. *The design and analysis of a computational model of cooperative coevolution*. PhD thesis, Citeseer, 1997.

[61] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[62] Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research (JAIR)*, 19:569–629, 2003.

[63] A. Pugh. *An introduction to tensegrity.* Univ of California Press, 1976.

[64] Schleip R, T. W. Findley, L. Chaitow, and P. Huijing, editors. *Fascia: The Tensional Network of the Human Body: The science and clinical applications in manual and movement therapy, 1e.* Churchill Livingstone, 1 edition, April 2012.

[65] John Rieffel, Davis Knox, Schuyler Smith, and Barry Trimmer. Growing and evolving soft robots. *Artificial Life*, 20(1):143–162, January 2014.

[66] John Rieffel, Francisco Valero-Cuevas, and Hod Lipson. Automated discovery and optimization of large irregular tensegrity structures. *Computers & Structures*, 87(5):368–379, 2009.

[67] John A. Rieffel, Francisco J. Valero-Cuevas, and Hod Lipson. Morphological communication: exploiting coupled dynamics in a complex mechanical structure to achieve locomotion. *Journal of the Royal Society Interface*, 7:613–621, 2009.

[68] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[69] T. Sandholm and R. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147–166, 1995.

[70] M. Shibata and S. Hirai. Moving strategy of tensegrity robots with semiregular polyhedral body. In *Proc. of the 13th Int. Conf. Climbing and Walking Robots (CLAWAR 2010), Nagoya*, pages 359–366, 2010.

[71] M. Shibata, F. Saijyo, and S. Hirai. Crawling by body deformation of tensegrity structure robots. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4375–4380. IEEE, 2009.

[72] R. E. Skelton and M. C. De Oliveira. *Tensegrity Systems.* Springer, 2009 edition, June 2009.

[73] Kenneth Snelson. Continuous tension, discontinuous compression structures. united states patent 3169611, Feburary 1965.

[74] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[75] Vytas SunSpiral, George Gorospe, Jonathan Bruce, Atil Iscen, George Korbel, Sophie Milam, Adrian Agogino, and David Atkinson. Tensegrity based probes for planetary exploration: Entry, descent and landing (EDL) and surface mobility analysis. *International Journal of Planetary Probes*, July 2013.

[76] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[77] Ming Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *ICML*, pages 330–337, 1993.

[78] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[79] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, MA, 2005.

[80] Tibert and et al. Review of form-finding methods for tensegrity structures. *International Journal of Space Structures*, 18:209–223, 2003.

[81] AG Tibert and Sergio Pellegrino. Review of form-finding methods for tensegrity structures. *International Journal of Space Structures*, 18(4):209–223, 2003.

[82] Brian R Tietz, Ross W Carnahan, Richard J Bachmann, Roger D Quinn, and Vytas SunSpiral. Tetraspine: Robust terrain handling on a tensegrity robot using central pattern generators. In *AIM*, pages 261–267, 2013.

[83] K. Tumer and A. K. Agogino. Multiagent learning for black box system reward functions. *Advances in Complex Systems*, 12:475–492, 2009.

[84] Kagan Tumer and Adrian Agogino. Improving air traffic management with a learning multiagent system. *IEEE Intelligent Systems*, 24:18–21, January 2009.

[85] J. M. Mirats Tur and S. H. Juan. Tensegrity frameworks: Dynamic analysis review and open problems. *Mechanism and Machine Theory*, 44:1–18, 2009.

[86] Bryan Wagenknecht and Dimitrios (Dimi) Apostolopoulos. Locomotion strategies and mobility characterization of a spherical multi-legged robot. In *Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2010,August 15-18, 2010, Montreal, Quebec, Canada*, number CMU-RI-TR-, August 2010.

[87] N. Wang, K. Naruse, D. Stamenović, J. J. Fredberg, S. M. Mijailovich, I. M. Tolić-Nørrelykke, T. Polte, R. Mannix, and D. E. Ingber. Mechanical behavior in living cells consistent with the tensegrity model. *PNAS*, 98(14):7765–7770, Jul 2001.

[88] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1st edition, 2000.

[89] R Paul Wiegand. *An analysis of cooperative coevolutionary algorithms*. PhD thesis, Citeseer, 2003.

[90] R Paul Wiegand, William C Liles, and Kenneth A De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, volume 2611, pages 1235–1245, 2001.

[91] R Paul Wiegand, William C Liles, and Kenneth A De Jong. Analyzing cooperative coevolution with evolutionary game theory. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1600–1605. IEEE, 2002.

[92] R Paul Wiegand and Mitchell A Potter. Robustness in cooperative coevolution. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 369–376. ACM, 2006.

[93] Xian Xu and Yaozhi Luo. Form-finding of nonregular tensegrities using a genetic algorithm. *Mechanics Research Communications*, 37(1):85–91, 2010.

[94] J. Y. Zhang and M. Ohsaki. Adaptive force density method for form-finding problem of tensegrity structures. *International Journal of Solids and Structures*, 43:5658–5673, 2006.

[95] Li Zhang, Bernard Maurin, and Rene Motro. Form-finding of nonregular tensegrity systems. *Journal of Structural Engineering*, 132(9):1435–1440, 2006.